ProDAQ

User Manual

ProDAQ 3080

Gigabit Ethernet VXIbus Slot-0 Interface

PUBLICATION NUMBER: 3080-XX-UM-1010

bustec

Bustec Production, Ltd.
World Aviation Park, Shannon, Co. Clare, Ireland
Tel: +353 (0) 61 707100, FAX: +353 (0) 61 707106

PROPRIETARY NOTICE

Table of Contents

Table of Figures

Chapter 1 - Introduction

1.1 Overview

The ProDAQ 3080 Gigabit Ethernet VXIbus Slot-0 Interface provides access to VXIbus instruments through a standard Gigabit LAN interface using the VXI-11 protocol. It is designed to function as a bridge between the established, time-tested and proven base of VXIbus instruments and the IEEE 802 Ethernet, which allows you to build any size of test and measurement system simply by connecting the instruments via standard LAN to your computer.

The ProDAQ 3080 provides a standardized Gigabit LAN interface with support for the VXI-11 protocol and an embedded WEB interface. It utilizes the new Tundra Tsi148 bridge to support the 2eVME block transfers specified in the revision 3.0 of the VXI standard in addition to all standard transfer modes. This allows for high-speed data transfers while maintaining backward compatibility to existing VXI rev. 1.3, 1.4 and 2.0 instruments.

The ProDAQ 3080 is fully compliant to the VXI*plug*&*play* standard. Access to the 3080 and the VXIbus instruments is provided through a standard VISA library. This allows for backward compatibility with existing VXI*plug*&*play* drivers and application software. The VXIbus resource manager is embedded in the 3080 firmware and automatically executed at power-up. The embedded WEB interface allows configuring and controlling the ProDAQ 3080 VXIbus Gigabit LAN Slot-0 interface and provides access to the VXIbus instruments via a standard WEB browser.

Communication with the host processor via the front-panel Gigabit Ethernet port is done via standard Cat 5e Ethernet cable for distances up to 200 meters. Low-cost Gigabit Ethernet switches can be used to increase the maximum distance as well as to connect multiple mainframes to a single host or to integrate multiple mainframes and hosts into a network.

Note:

To achieve maximum performance, connect the ProDAQ 3080 to a host featuring a Gigabit LAN interface. If you are using switches or hubs in your network connection, make sure that they conform to the Gigabit Ethernet standard and are able to operate at that speed.

For synchronization in legacy systems, the ProDAQ 3080 features a front-panel trigger input/output and CLK10 I/O via SMB connectors.

This page was intentionally left blank.

Chapter 2 - Getting Started

The ProDAQ 3080 Gigabit Ethernet VXIbus Slot-0 Interface is a single slot, C-size VXIbus instrument and can be installed in any slot of a standard C-size VXI mainframe. To be Slot-0 controller for the VXIbus system, it must be installed in the leftmost slot of the VXI mainframe (slot “0“). If it is installed in any other slot of a VXI mainframe, all slot-0 capabilities (MODID, CLK10, etc.) will be automatically turned off.

Attention:

To allow access to instruments in the VXI mainframe, the ProDAQ 3080 MUST be installed in the leftmost slot of the VXI mainframe (slot "0"). Installing it into any other slot will only allow you to access the device itself (e.g. for configuration purposes).

Installing it into any other slot will only allow you to access the device itself (e.g. for configuration purposes). If you do so, please make sure to set up the logical address correctly to avoid any collision with a slot-0 device already present in the mainframe.

Note:

The ProDAQ 3080 Gigabit Ethernet VXIbus Slot-0 Interface does not extend the VXI backplane between mainframes in a multi-mainframe system. This means that devices sharing the local bus must be installed in the same mainframe.

To install the ProDAQ 3080 Gigabit Ethernet VXIbus Slot-0 Interface and the necessary software on your system, use the installation sequence as described in this chapter:

- Step 1: Unpacking and Inspection
- Step 2: Installing the ProDAQ 3080
- Step 3: Connecting the ProDAQ 3080 Interface
- Step 4: Installing the VISA Library
- Step 5: Accessing the ProDAQ 3080

2.1 Unpacking and Inspection

All ProDAQ modules are shipped in an antistatic package to prevent any damage from electrostatic discharge (ESD). Proper ESD handling procedures must always be used when packing, unpacking or installing any ProDAQ module, ProDAQ plug-in module or ProDAQ function card:

- Ground yourself via a grounding strap or similar, e.g. by holding to a grounded object.
- Remove the ProDAQ module from its carton, preserving the factory packaging as much as possible.
- Discharge the package by touching it to a grounded object, e.g. a metal part of your VXIbus chassis, before removing the module from the package.

- Inspect the ProDAQ module for any defect or damage. Immediately notify the carrier if any damage is apparent.
- Only remove the module from its antistatic bag if you intend to install it into a VXI mainframe or similar.

When reshipping the module, use the original packing material whenever possible. The original shipping carton and the instrument's plastic foam will provide the necessary support for safe reshipment. If the original anti-static packing material is unavailable, wrap the ProDAQ module in anti-static plastic sheeting and use plastic spray foam to surround and protect the instrument.

2.2 Installing the ProDAQ 3080 Interface

To prevent damage to the ProDAQ module being installed, it is recommended to remove the power from the mainframe or to switch it off before installing.

2.2.1 Configuring the Logical Address

To allow a host to control the VXI devices in the mainframe via the network using the ProDAQ 3080, the ProDAQ 3080 must be installed as the slot-0 controller for the mainframe, i.e. it must be installed in the leftmost slot of the mainframe (slot "0") and must be configured for using logical address 0 (zero).

The logical address switch is located on the back of the module. Figure 1 shows the location of the logical address switch on the ProDAQ 3080. Set each switch to 'Off' for a logical one (1) and to 'On' for a logical zero (0). The picture shows the address switch set to logical address zero (0).

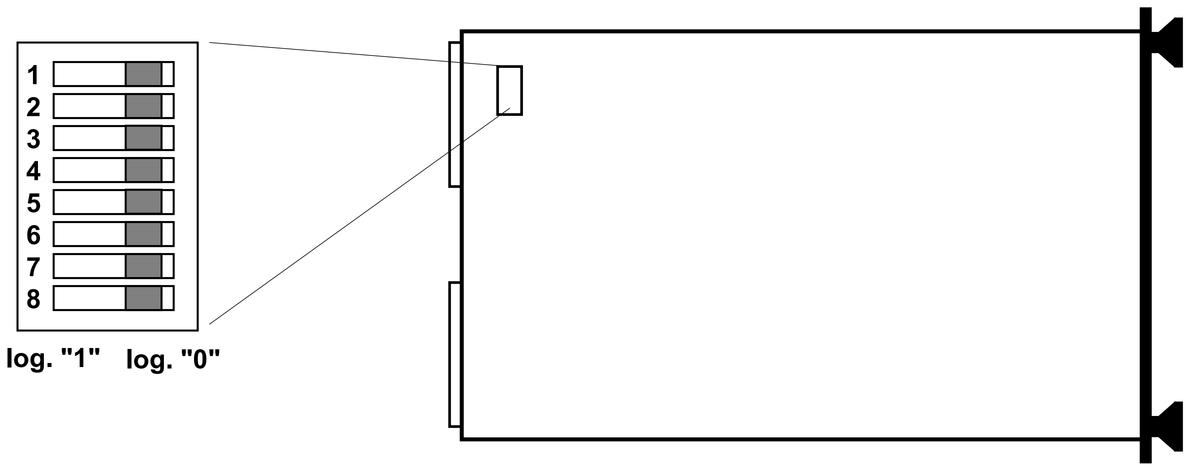

Figure 1 - Logical Address Switch Location

2.2.2 Installing the ProDAQ 3080 into the Mainframe

Insert the module into the mainframe using the guiding rails inside the mainframe as shown in Figure 2. Push the module slowly into the slot until the backplane connectors of the module seat firmly in the corresponding backplane connectors. The top and bottom of the front panel of the module should touch the mounting rails in the mainframe.

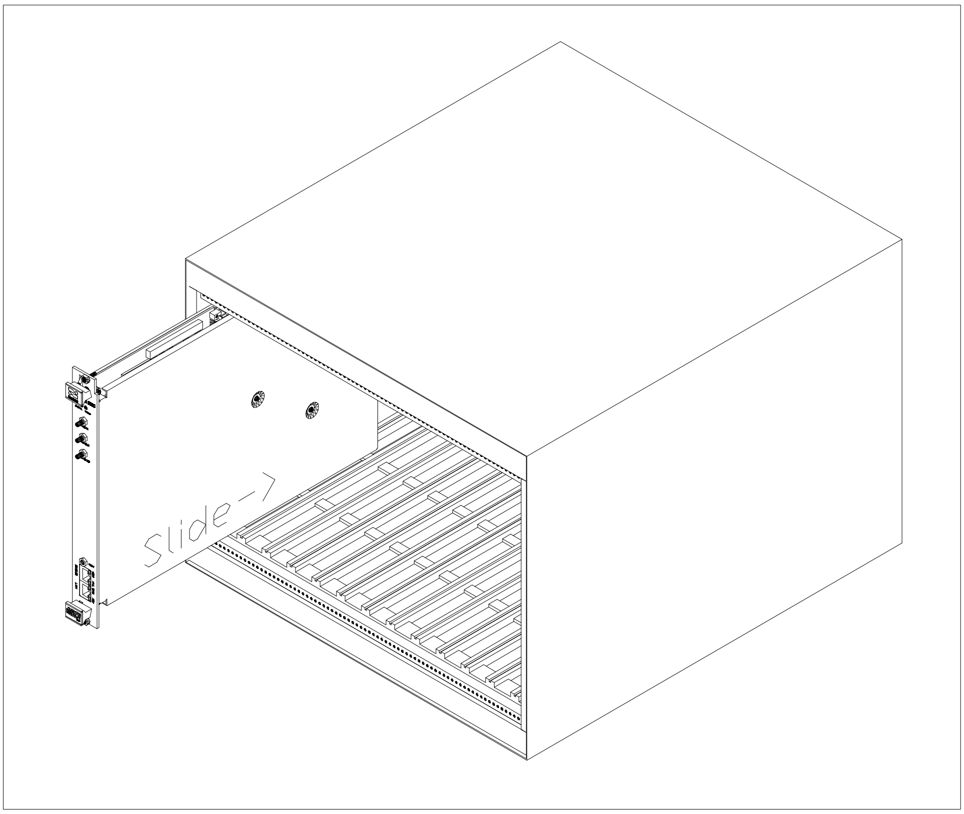

Figure 2 - Installing the ProDAQ 3080 into a C-Size Mainframe

---

Note:

To ensure proper grounding of the module, tighten the front panel mounting screws after installing the module in the mainframe.

2.3 Connecting the ProDAQ 3080 Interface

The ProDAQ 3080 is equipped with a standard RJ-45 network connector, accepting standard Cat 3, Cat 5, Cat 5e and Cat 6 Ethernet cables. However, to run the interface in a network using 1000BASE-T mode, in minimum Cat 5e (better Cat 6) cables are required.

The figure on the next page shows the location of the LAN connector on the ProDAQ 3080 front panel. The connector features two LED indicators showing the speed and the link status of the connection made (see Table 1).

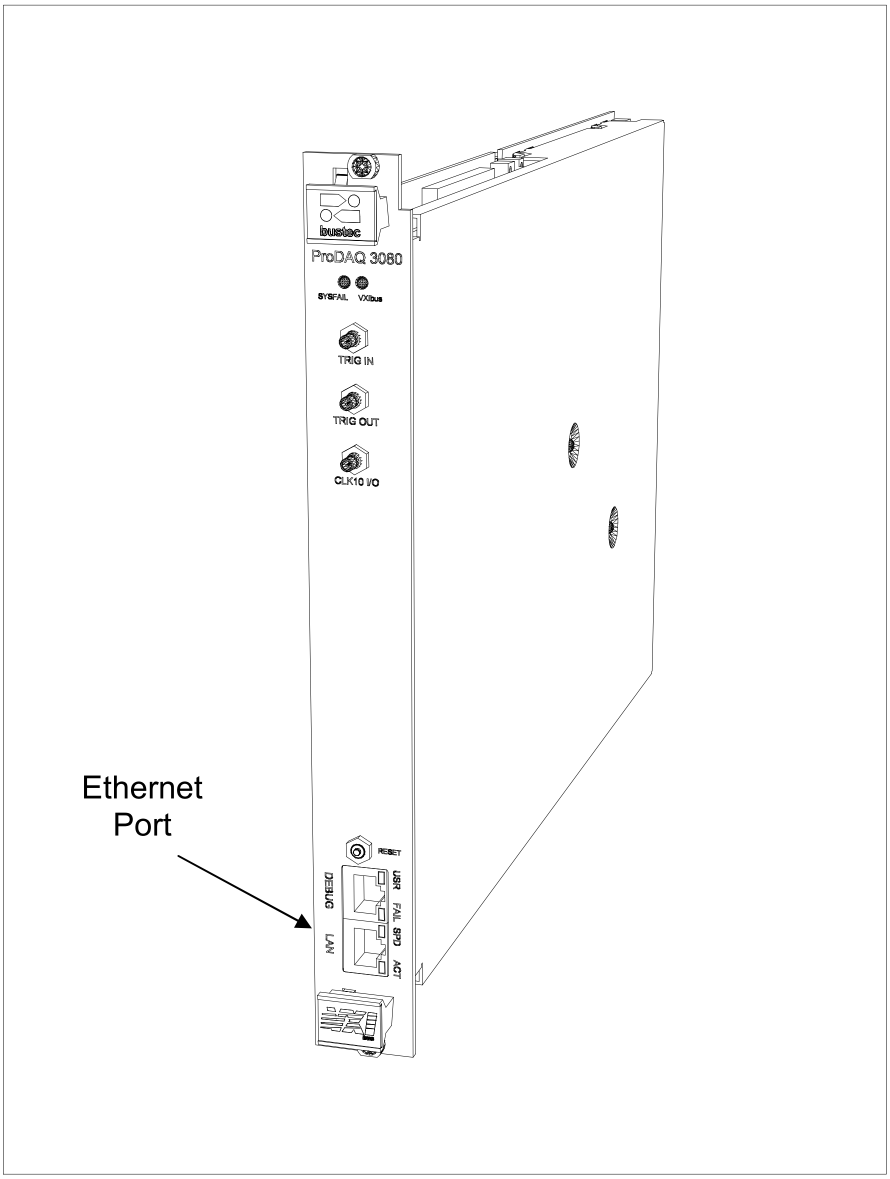

Figure 3 - ProDAQ 3080 Ethernet Port

LED	**Color**	**Description**
SPEED	Off	No link
	Yellow	10BASE-T/100BASE-T operation
	Green	1000BASE-T operation
ACT	Off	No Activity
	Blinking Green	Activity proportional to bandwidth utilization.

Table 1 - LAN Status Indicators

2.4 Installing the VISA Library

The VISA library provided by Bustec Production Ltd is used to communicate to the ProDAQ 3080 and the VXI instruments installed together with the ProDAQ 3080 in the same mainframe.

Note

On Microsoft Windows 2000® or Microsoft Windows XP® systems it is recommended to install the VISA library from an account having administrator privileges.

To install it on your PC, do the following:

1. Apply power to your PC and boot your operating system. Close all open applications to allow for a safe installation of the new components.

2. Insert the driver CD provided with the module into your PC CD-ROM drive. If the autorun feature is turned on, the CD menu will start automatically. If not, select "Run" from your Start menu and type <drive>:autorun.exe, where <drive> designates the CD-ROM drive with the driver CD in it.

3. Select "VISA Library for ProDAQ Controller" from the driver section of the CD menu to start the setup wizard.

   Please note: If you have downloaded the Bustec VISA Library from our WEB site, all files are packed into a single ZIP archive. To start the installation, unpack the files into a separate directory on your drive and run the executable "setup.exe" from that location.

4. Select "Next" to review the license agreement for the Bustec VISA library. You will need to accept the terms of the agreement by selecting "Yes" to be able to install the Visa library.

5. Select the folder where the wizard will install the components of the VISA library. Please note that the location chosen will be the top-level directory for a VXI*plug*&*play* standard compliant directory tree, and not a single location for the library only. If you install VXI*plug*&*play* driver on your PC, they will install using the directory tree created by the VISA installation.

6. Select "Next" to choose the type of setup to perform (see Figure 4). "Typical" will install the most common components, while "Compact" will only install the absolute necessary components. To choose which components to install, choose "Custom".

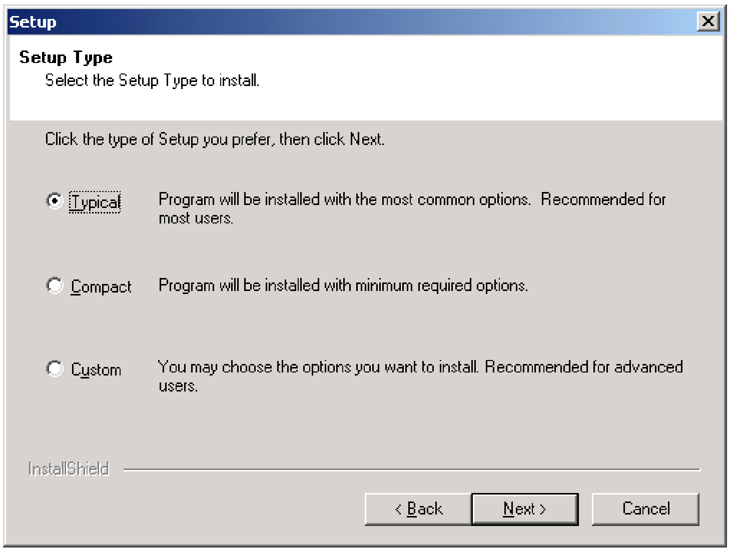

Figure 4 - Selecting the Type of Installation.

7. If you have chosen “Custom”, selecting “Next” will allow you to select the components to install (see Figure 5):

VISA Library	The core files (hardware driver, VISA dynamic link library, config utility, include files) of the installation.
VISA Assistant	An interactive graphical user interface for the VISA library. It will allow you to use the VISA library without writing your own application.
Help Files	Help files for the VISA library.
Examples	How to program using the VISA library.

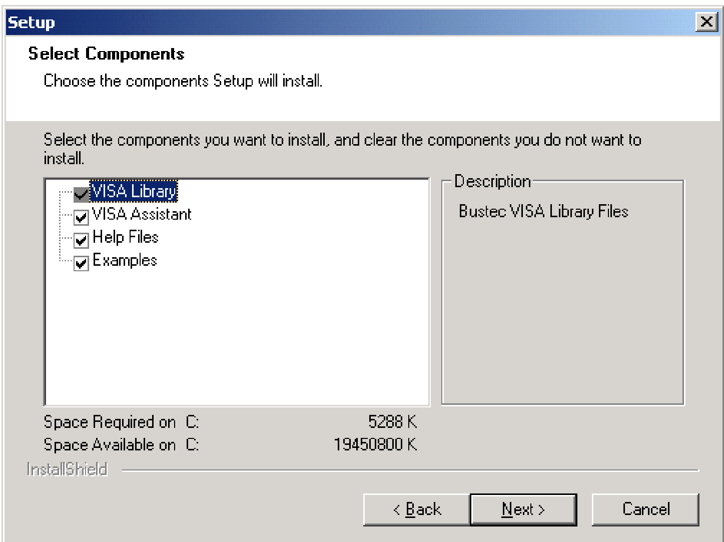

Figure 5 - Selecting Components for Installation.

8. After selecting “Next”, the wizard will install the files and components for the chosen configuration on your system.

9. The next dialog allows you to select options for installing shortcuts to the resource manager, configuration utility and the VISA assistant on your desktop as well as to install a shortcut to the resource manager in the “Startup” folder, which will cause the resource manager to be run automatically when the system boots.

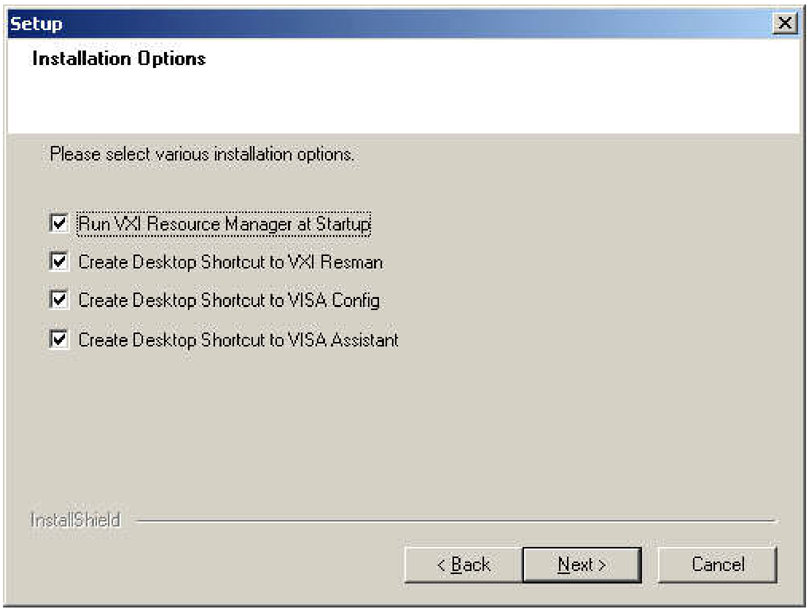

Figure 6 - Selecting Installation Options

10. After selecting next, the installation is complete. Please choose whether you want to view the readme for the VISA distribution now or whether you want to run the configuration utility immediately to complete the configuration and click “Finish”.

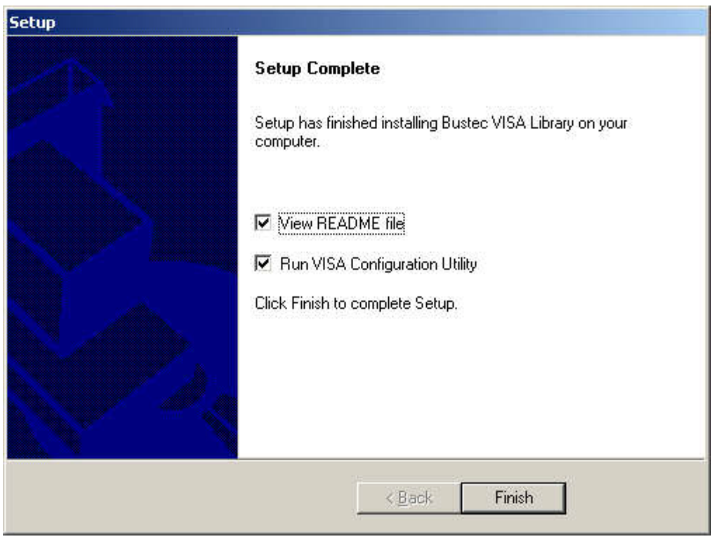

Figure 7 - Finishing the Setup

11. Re-start the computer after the installation is complete.

2.5 Accessing the ProDAQ 3080

By default the ProDAQ 3080 uses DHCP to configure its network interface. If no DHCP server is found in the network, it will attempt to obtain a network address using AutoIP. AutoIP addresses are allocated from the reserved range 169.254.0.0 -169.254.255.255. The ProDAQ 3080 will first try to use the address 169.254.x.y, where <x> and <y> are the two last octets of the devices MAC address. If the address is already in use, a new pair of <x> and <y> will be generated using a random number generator. By using the embedded web interface, the ProDAQ 3080 can also be configured to use a static IP address.

2.5.1 Accessing the ProDAQ 3080 using Dynamic DNS

If there is a Dynamic DNS server available in the network, the instrument can be accessed via its hostname. The default hostname is:

prodaq3080-<*serial number*>.<*domain*>

Where:

<*serial number*> is the 8-digit serial number of the device,
<*domain*> is defined by the Dynamic DNS server.

To access the instruments WEB pages, just enter the hostname into the address bar of your internet browser, for example:

http://prodaq3080-10478812.local <***return***>

The serial number of the device can be found on the product label on the module cover.

2.5.2 Discovering the ProDAQ 3080 using Multicast DNS

The ProDAQ 3080 publishes the availability of its HTTP service, so that clients using the Multicast DNS protocol and DNS Service Discovery can access it without knowing its actual IP address. Using a zero-configuration networking tool like for example Bonjour for Windows (for Microsoft Windows 32-bit and 64-bit Operating Systems, available at www.apple.com) or Avahi (for 32-bit or 64-bit Linux, available at www.avahi.org) will allow you to browse your network and discover all ProDAQ 3080 devices. The default mDNS service name is:

Bustec Production Ltd - ProDAQ3080 - <*serial number*>

Where:

<*serial number*> is the 8-digit serial number of the device.

To access the instrument using Bonjour, open the Internet Explorer on your host computer and check whether the Bonjour explorer bar is visible. If not, use the Bonjour icon or the menu entry (“View” → “Explorer Bars” → “Bonjour”) to show it. Double click the ProDAQ 6100 entry to open the instruments embedded web page (see Figure 8).

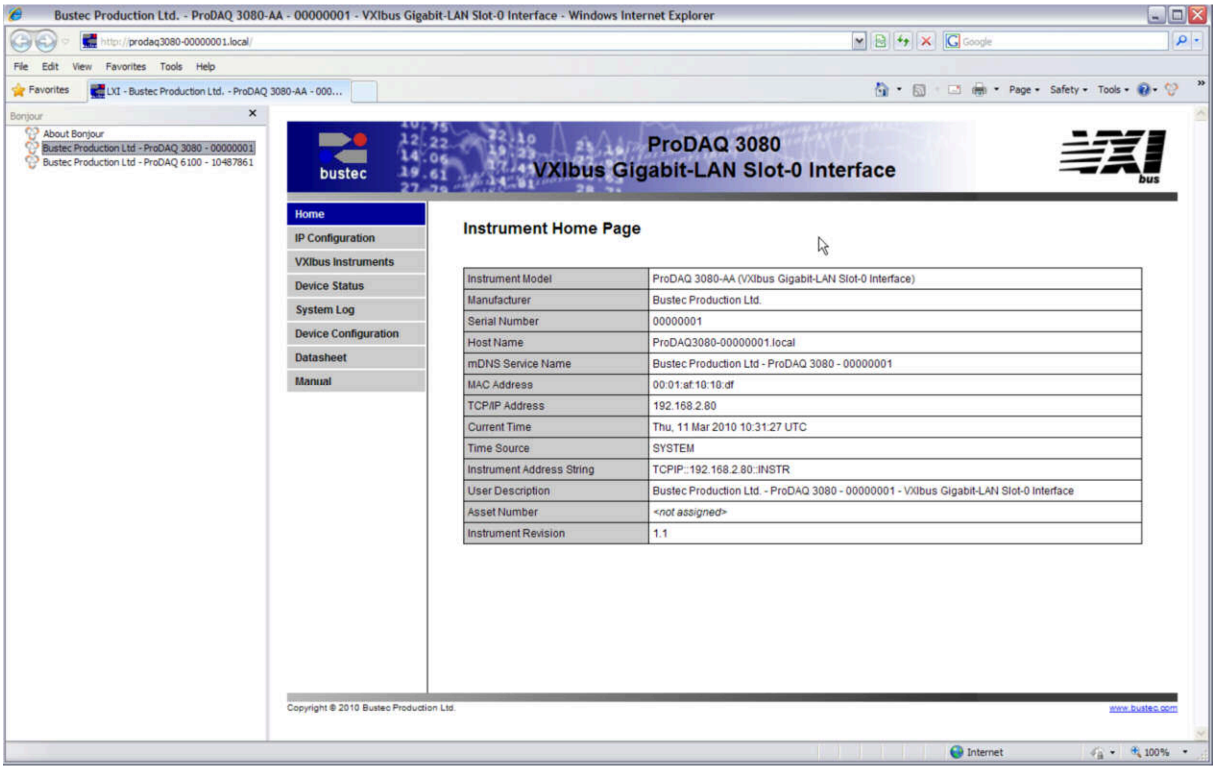

Figure 8 - Using Bonjour to discover the ProDAQ 3080

2.5.3 Discovering the ProDAQ 3080 using VXI-11 Broadcast

The configuration utility coming with the Bustec VISA can be used to discover the ProDAQ 3080 using the VXI-11 protocol. To start the application select the “VISA Configuration Utility” (“Start” → “VXIPNP” → “VISA Configuration Utility”) from the VXI*plug*&*play* program group created during the installation of the VISA library. This will start the configuration tool for the VISA library and attached hardware interfaces.

Select the tab “Network Instruments” and select the button “Find Instruments…” to the right of the list of Instrument Descriptors. This will open the “Find Network Instruments” dialog. Here you can select the device type of network instrument to discover and the range of interface numbers.

To discover any ProDAQ 3080 located in your subnet, at least the interface device type “VXI” needs to be enabled with a range starting with zero (“0”). To start the search, select the button “Start Searching” to the right. After the search is complete, the list below will show the network instruments found (see Figure 9).

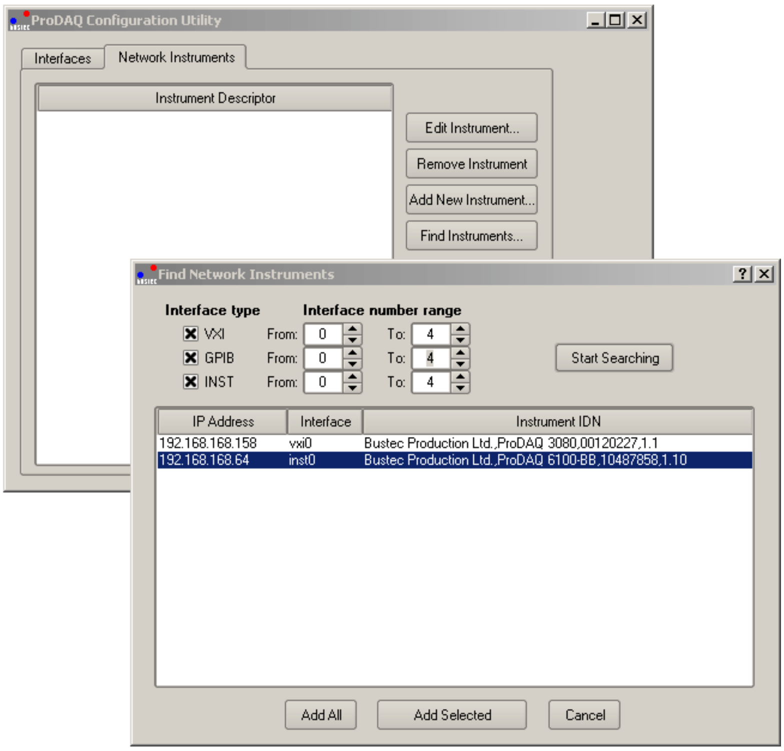

Figure 9 - ProDAQ Configuration Utility

Chapter 3 - WEB Page Operation

The ProDAQ 3080 features an embedded WEB server, which allows you to configure and operate the ProDAQ 3080 by using a standard WEB browser from any host computer in your network.

3.1 Instrument Home Page

The instrument home page shows general information about the device like model number, manufacturer, serial number, and revisions.

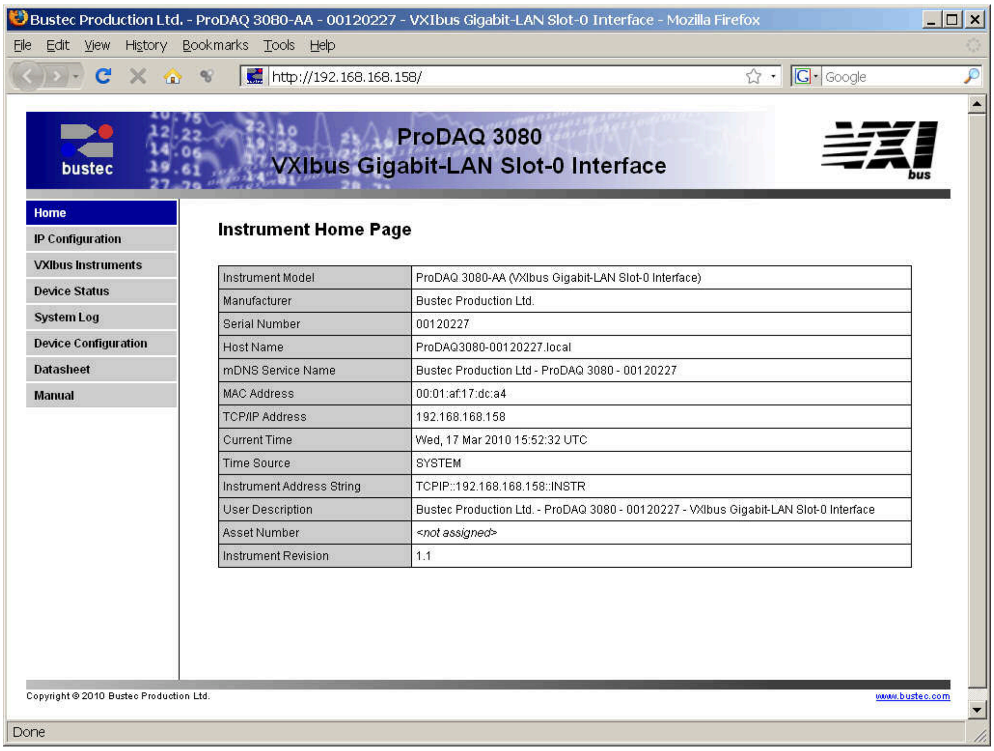

Figure 10 - Instrument Home Page

From here you can navigate to the different categories and pages by using the menu on the left side.

For security reasons, all pages except of the instruments home page are protected by username and password, which can be configured on the "Device Configuration" -> "Security Settings" page. Upon delivery, the username and is set to "admin" and the password to "1234".

3.2 IP Configuration

The IP Configuration Page allows you to change the settings for the ProDAQ 3080's LAN interface.

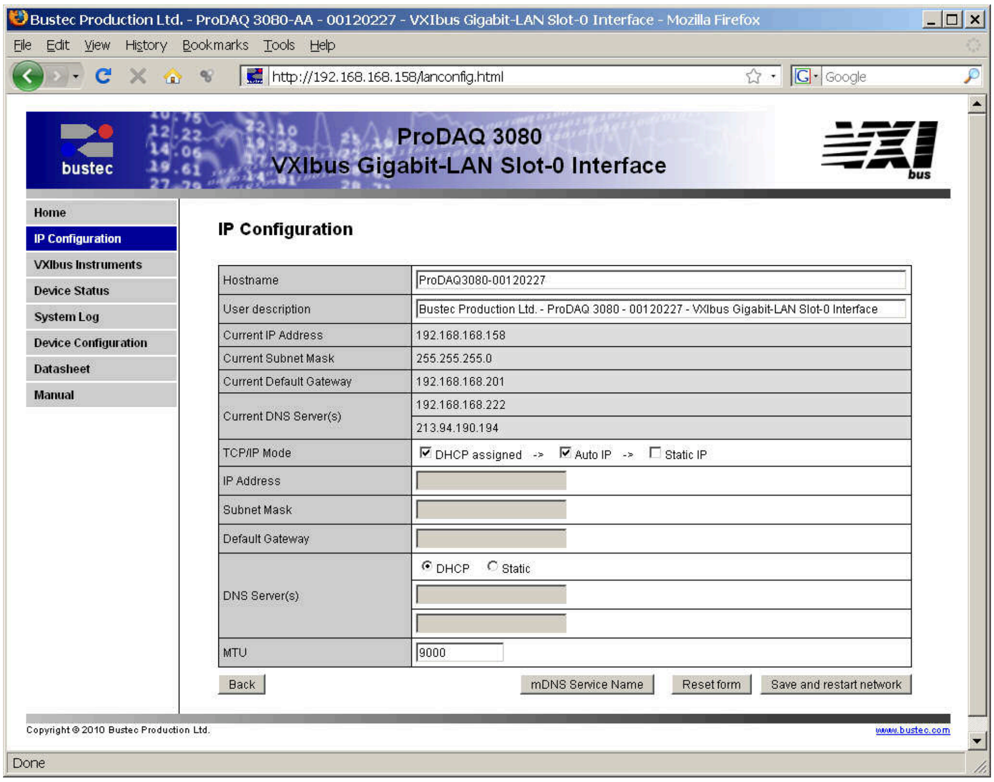

Figure 11 - IP Configuration Page

The IP Configuration page shows the current settings for the instruments LAN interface and allows you to change and store the following settings:

Hostname

User defined hostname for the device (without domain).
Clear this value to revert to factory default.

Note: Multicast DNS domain is always: ".local". Dynamic DNS domain depends on the network configuration.

User Description

User defined description of the device – it is displayed on the Home Page along with user defined Asset Number (see Device Configuration).

Clear this value to revert to factory default.

Current IP configuration

Displays currently assigned: IP Address, Subnet Mask, Default Gateway and DNS servers.

TCP/IP mode	Specifies whether the device shall use a DHCP server in the network, or AutoIP protocol to automatically obtain the IP configuration, or maybe the static IP configuration defined in the form below. More than one option may be selected. The priority is as follows: DHCP → AutoIP → Static. For example, if DHCP and Static are selected and DHCP fails, the Static configuration is set.
IP Address	If "Static IP" was selected as the TCP/IP mode, this field allows assignment of a static IP address to the ProDAQ 6100s LAN interface.
Subnet mask	If "Static IP" was selected as the TCP/IP mode, this field allows assignment of a static subnet mask address to the ProDAQ 6100s LAN interface.
Default Gateway	If "Static IP" was selected as the TCP/IP mode, this field allows assignment of a static default gateway for the routing of IP packets.
DNS Servers	If "Static IP" was selected as the TCP/IP mode, these two fields allow you to specify the DNS server the ProDAQ 6100 will use for name resolving. If “DHCP” was selected as the TCP/IP mode, then it is possible to select whether the DNS servers’ IP addresses shall be acquired automatically (DHCP) or user-defined (Static).
MTU	Maximum Transmission Unit (MTU) – maximum size (in bytes) of an IP packet that can be transmitted without fragmentation (including IP headers, but excluding headers from lower levels in the protocol stack). The default value for a typical network is 1500 bytes. It can be defined as high as 9000 bytes (jumbo frames). For correct interoperation, the whole network must have the same MTU. To achieve the maximum performance, it is recommended to configure the network to work with a MTU settings as high as possible.
mDNS Service Name	User defined name of mDNS services that are advertised by the ProDAQ 3080. Clear this value to revert to factory default.

3.3 VXIbus Instruments

The VXIbus Instruments Configuration page shows a table with the VXIbus instruments identified by the embedded resource manager on start-up.

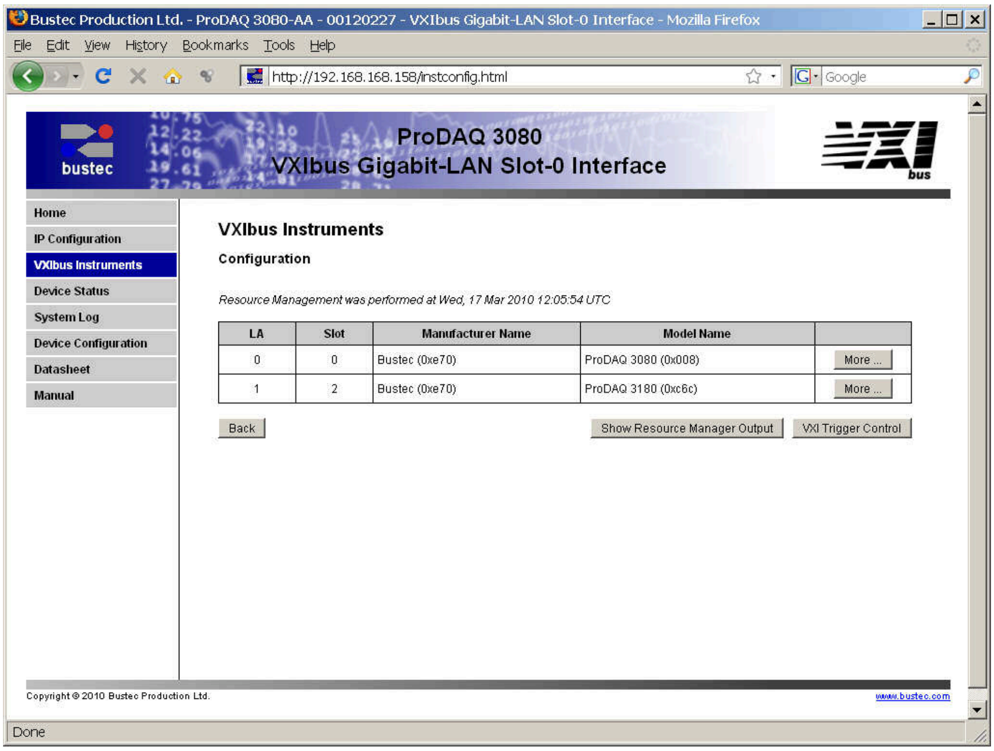

Figure 12 - VXIbus Instruments Page

By pressing the "More..." buttons to the right of an instruments entry, a separate page with additional information about the particular device is shown (see 3.3.1), where you can perform basic I/O operations in a way similar to the VISA assistant.

The "Show Resource Manager Output" button displays the log file written by the embedded resource manager on start-up.

The “VXI Trigger Control” button lets you access a page where you can set the routing of the VXIbus backplane trigger lines and the ProDAQ 3080 front panel trigger I/Os.

3.3.1 Instrument Information and Access

The “Instrument Information and Access” page shows detailed information about the VXIbus instrument as discovered by the embedded resource manager.

Figure 13 - Instrument Information and Access Page

Depending on the type of instrument you can perform basic memory or message based access operations on the device by selecting the “Memory I/O” or “Basic I/O” buttons at the bottom of the page.

Figure 14 - Instrument Memory I/O

3.3.2 Resource Manager Output

The “Resource Manager Output” page lets you review the output of the embedded VXIbus resource manager.

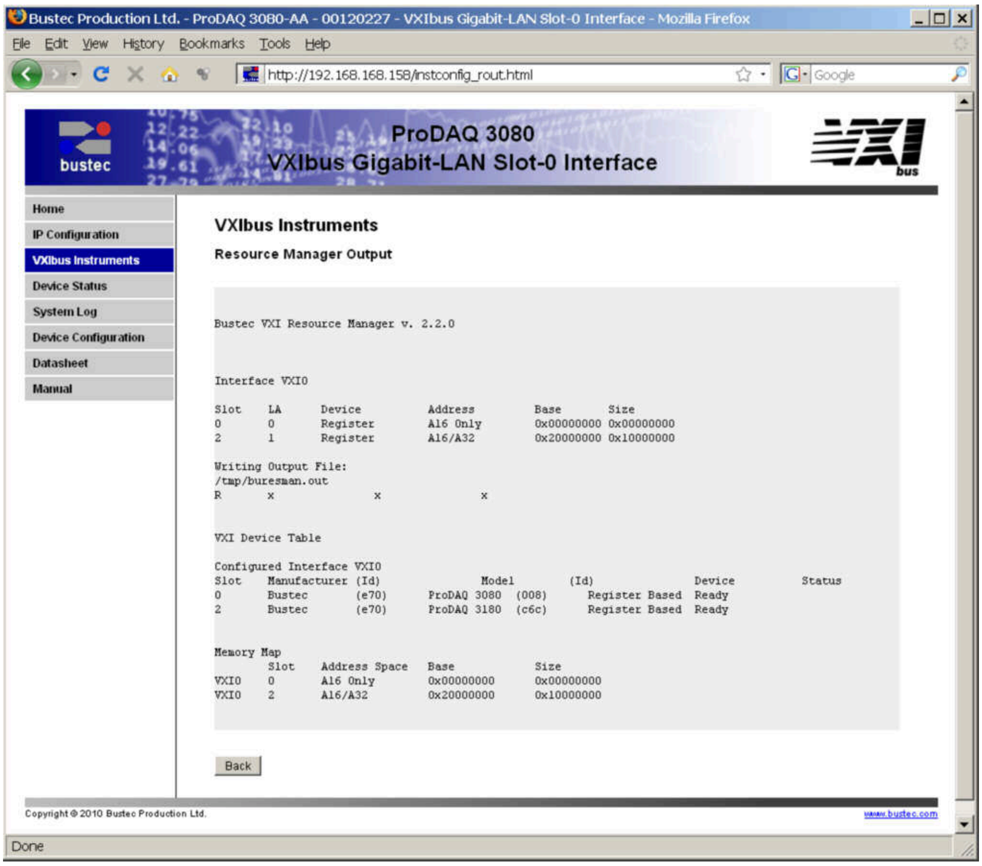

Figure 15 - Resource Manager Output Page

3.3.3 VXI Trigger Control

The “VXI Trigger Control” page allows you to route the VXIbus trigger lines from/to the front panel trigger I/Os on the ProDAQ 3080 (see Figure 16).

By choosing the “Front Panel Trigger In” selection for any or all of the VXIbus TTL and ECL trigger lines or the Front Panel Trigger Output line, any trigger received on the front panel trigger input line of the ProDAQ 3080 will be routed to any or all of the chosen lines.

While “Unrouted” and “Front Panel Trigger In” are the only possible sources for trigger events for the VXIbus TTL and ECL Trigger lines, the front panel trigger output line can also receive trigger events from the VXIbus TTL and ECL trigger lines.

In addition to routing the trigger lines, each of the trigger lines can be asserted, de-asserted or a pulse can be generated by using the buttons “Assert”, “Deassert” or “Pulse” to the right of each trigger line source selection.

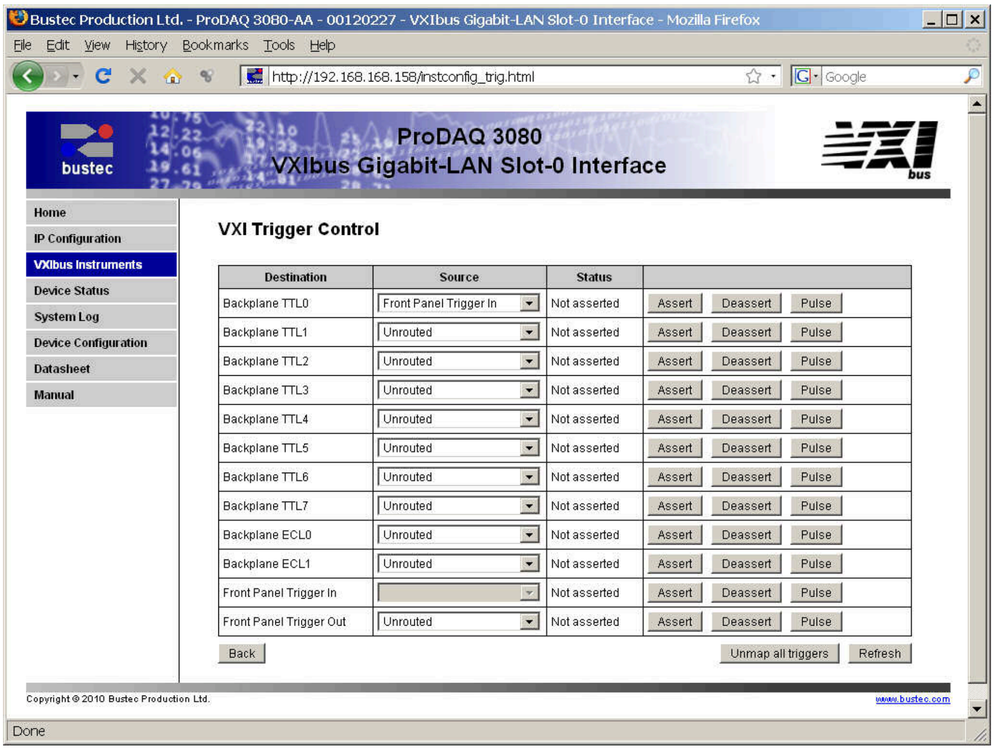

Figure 16 - VXIbus Trigger Control

3.4 Device Status

The “Device Status” page shows the overall status of the ProDAQ 3080 and its network connection. For a more detailed status, select the “Show advanced status” button at the bottom.

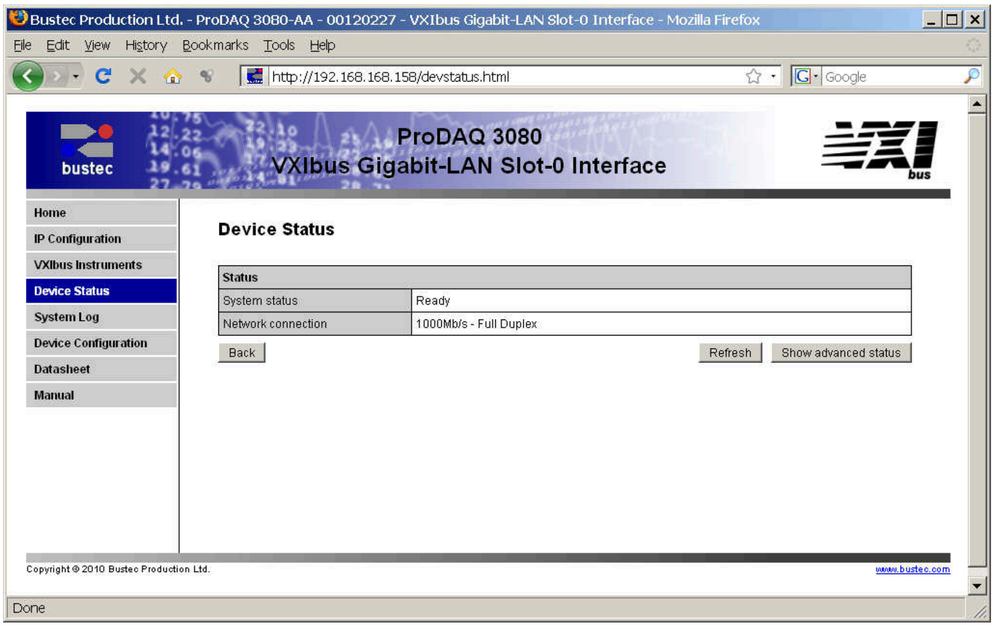

Figure 17 - Device Status Page

3.4.1 Advanced Status

The “Advanced Status” page allows you to view the output of several tools and contents of configuration files available on the ProDAQ 3080. To switch between the different outputs/files, just select the tool/file with the combo box at the bottom. The “Refresh” button allows updating the status.

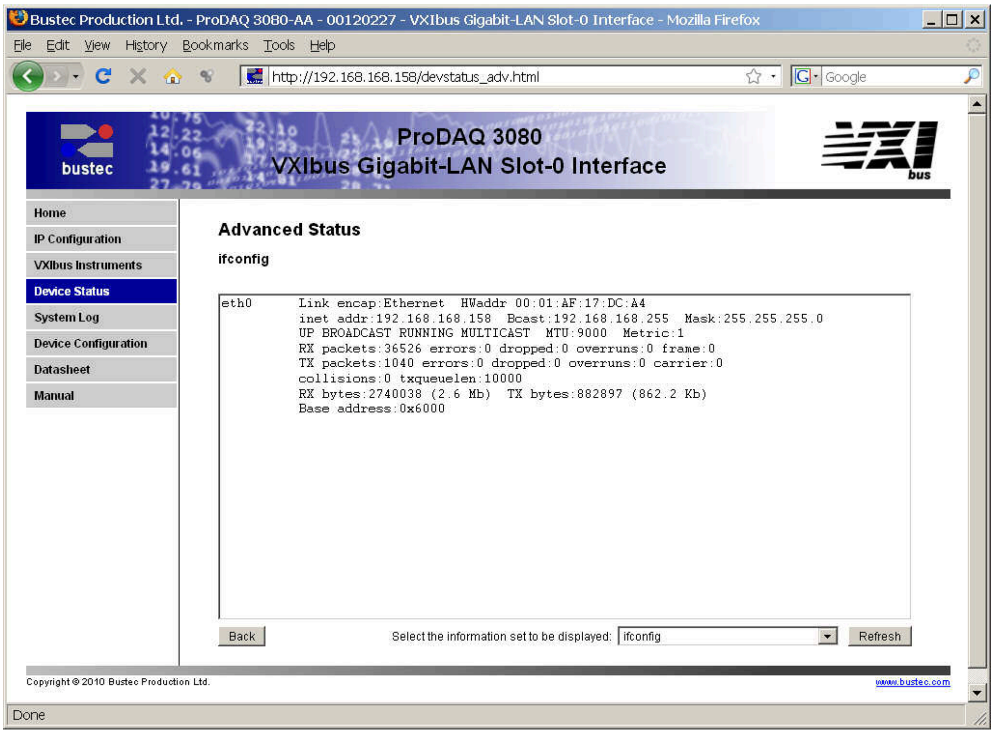

Figure 18 - Advanced Status Page

The following tools and configuration files are available:

ifconfig	Shows the output of the “ifconfig” utility with detailed information on the network interface status. For detailed information, please refer to the Linux manual page for “ifconfig”.
route	Shows the routing table as seen by the embedded Linux kernel on the ProDAQ 3080.
resolv.conf	Displays the contents of the *resolv.conf* file. The resolv.conf file is maintained by networking scripts and shows the current nameserver configuration in use by the ProDAQ 3080 kernel.
hosts	Displays the contents of the *hosts* file. The *hosts* file contains the known host aliases.
device.conf	The *device.conf* file holds the static settings configured via the “IP Setup” page.
device and firmware revision	Shows the revisions of the different parts of the system.

3.5 System Log

The “System Log” page shows the output of the kernel logging facility.

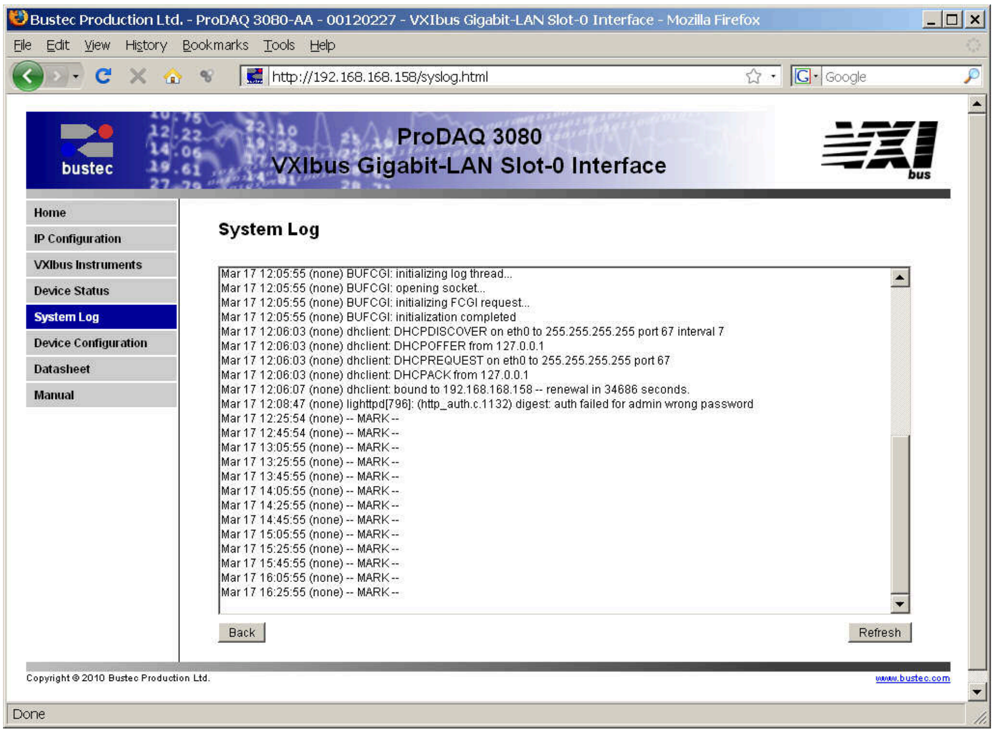

Figure 19 - System Log Page

3.6 Device Configuration

The device configuration is split up into several sub-items. Click on one of the buttons to the right of the different sections to access it. Each sub-item lets you configure a part of the ProDAQ 3080.

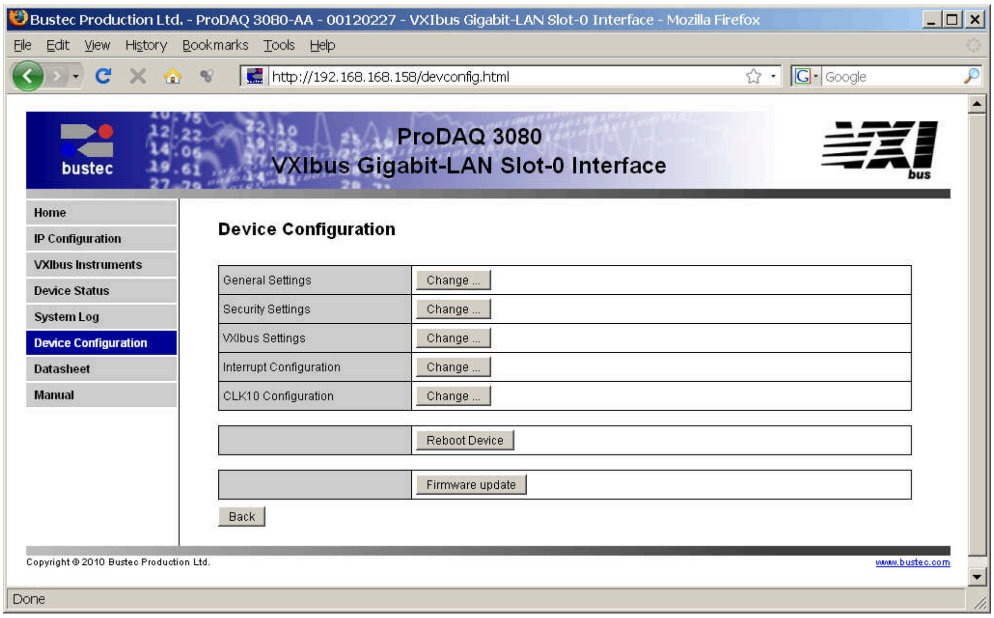

Figure 20 - Device Configuration Page

In addition the page contains two buttons to either reboot the device or to update the firmware.

3.6.1 General Settings

This page allows you to change the system time and assign an asset number to the device, which will be shown on the instrument home page.

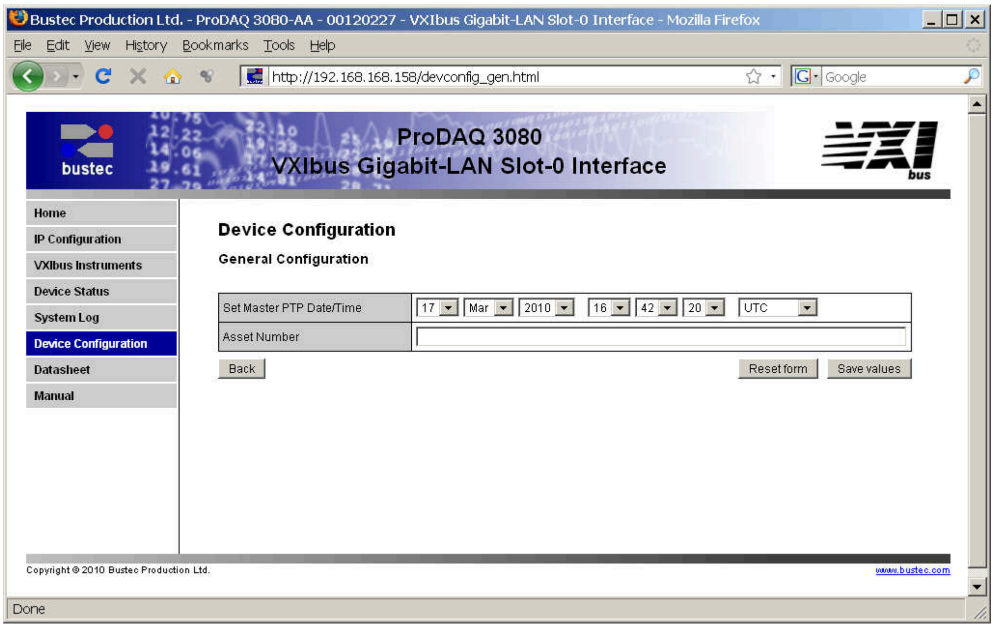

Figure 21 - General Configuration Page

3.6.2 Security Settings

On this page you can change the password that is used to protect the pages of the ProDAQ 3080. Please type in your old password, the new one and confirm it by re-typing.

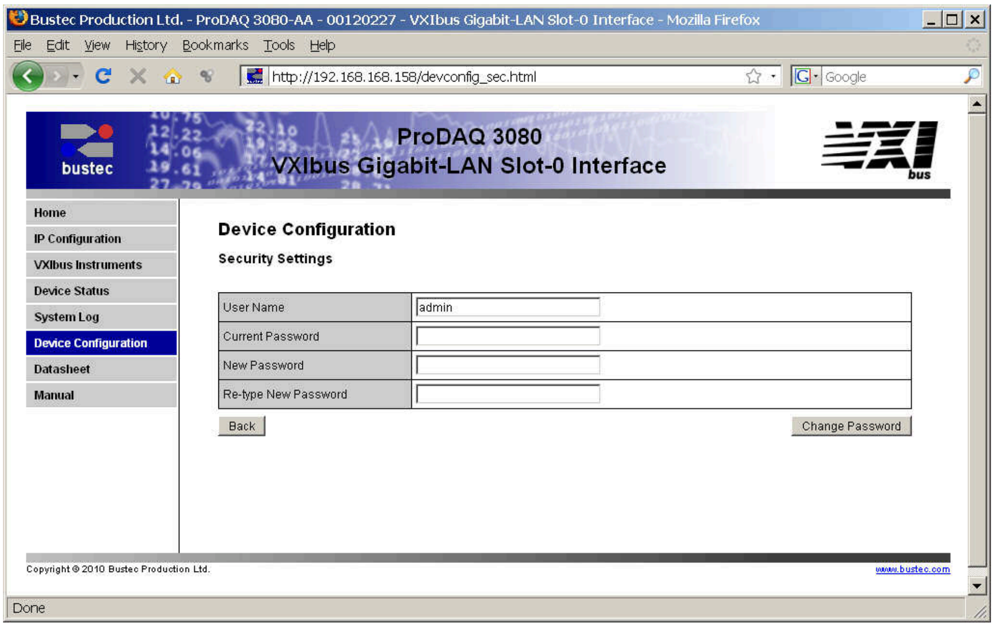

Figure 22 - Security Settings Page

3.6.3 VXIbus Settings

The VXIbus Settings page allows you to configure how the ProDAQ 3080 accesses the VXIbus.

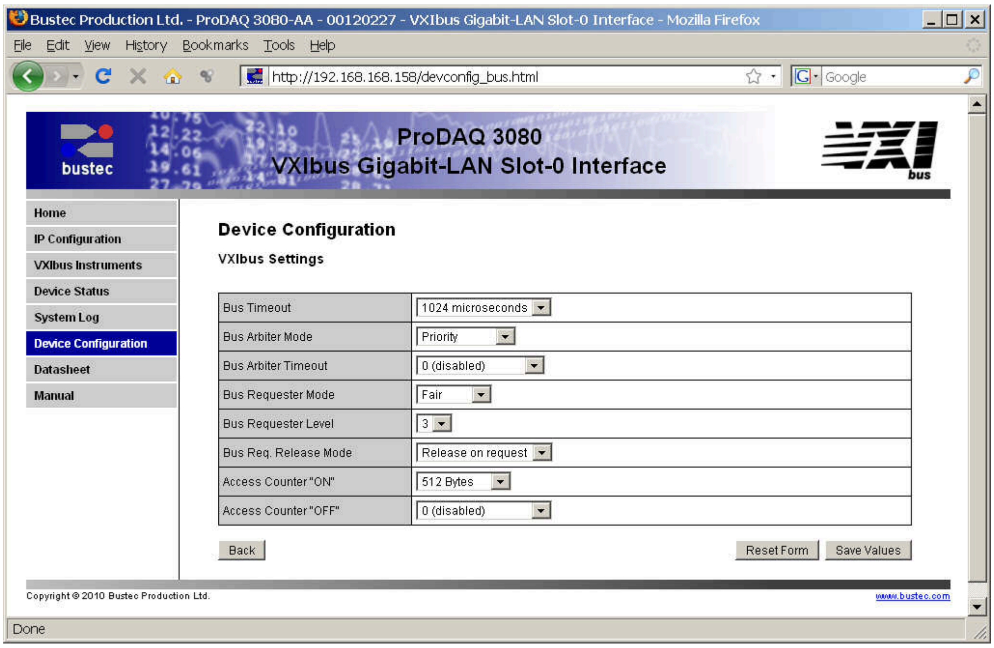

Figure 23 - VXIbus Settings Page

Bus Timeout	The time the on-board timer needs to expire once a VXIbus access by the 3080 is started. If it expires, a VXIbus slave did not respond correctly and a bus error is generated. Possible values are: Disabled, 16 µsec, 32 µsec, 64 µsec, 128 µsec, 256 µsec, 512 µsec and 1024 µsec
Bus Arbiter Mode	Selects the bus arbiter mode. Possible values are: “Priority” or “Round Robin”. (Remark: The arbiter is only enabled if the module is placed in the leftmost slot of a VXI mainframe, slot “0”).
Bus Arbiter Timeout	Specifies the timeout for the arbiter. Possible values are: Disabled, 16 µsec, 256 µsec.
Bus Requester Mode	Sets the request mode of the ProDAQ 3080, “Fair” or “Demand”.
Bus Requester Level	Selects the request level the module is using when accessing the VXIbus. Possible values are 3 to 0, with 3 as the highest priority and 0 as the lowest.

Bus Req. Release Mode	Selects the release mode: “RWD” (release when done) or “ROR” (release on request).
Access Counter "On"	Sets the number of bytes to transfer before a bus access can be interrupted. Possible values are 0 (disabled), 256 bytes, 1024 bytes, 2048 bytes, 4096 bytes, 8192 bytes and 16384 bytes.
Access Counter "Off"	Sets the time the accesses are paused before a new block is started. Possible values are: 0 (disabled), 2 µs, 4 µs, 8 µs, 16 µs, 32 µs, 64 µs, 128 µs, 256 µs, 512 µs and 1024 µs.

Note:

Please note that any changes will be applied only at the next reboot of the device.

3.6.4 Interrupt Configuration

The Interrupt Configuration page allows configuring the usage of the VXIbus interrupt lines in the allocation mechanism of the VXI resource manager.

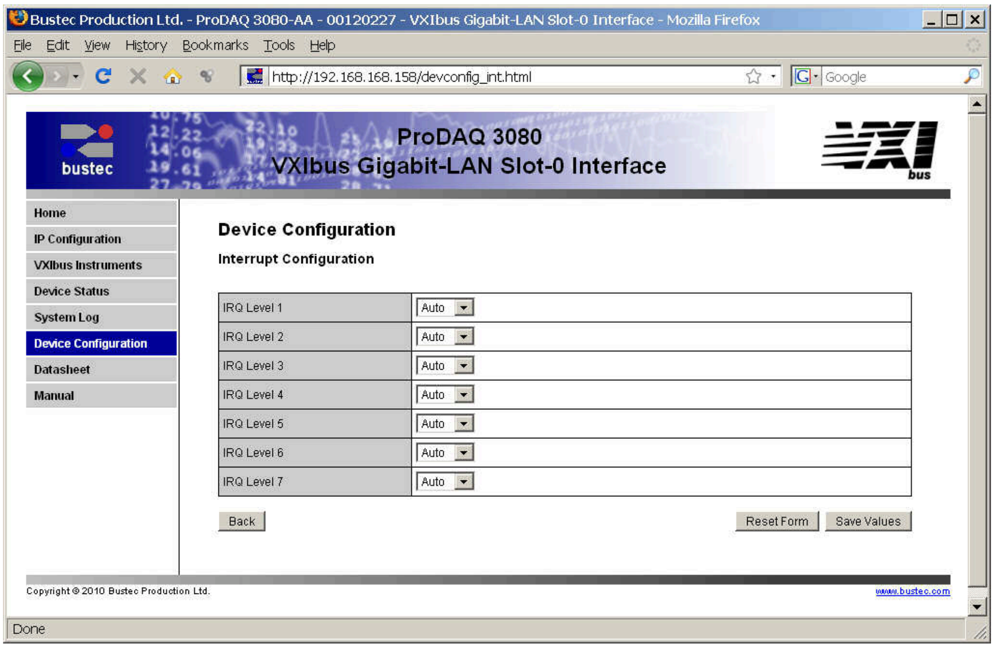

Figure 24 - Interrupt Configuration Page

For each of the VXIbus interrupt lines (Level 1 to Level 7) one of two settings for the assignment can be chosen:

Auto	This setting will allow the resource manager to use the interrupt line for this level in his allocation mechanism.

None — This setting will prevent the resource manage to use the interrupt line for this level in his allocation mechanism. This setting must be used if a instrument in the system does not allow the dynamic allocation of interrupt lines and wants to use one or more lines permanently allocated.

3.6.5 CLK10 Configuration

This page allows you to enable or disable the CLK10 output on the ProDAQ 3080 front panel.

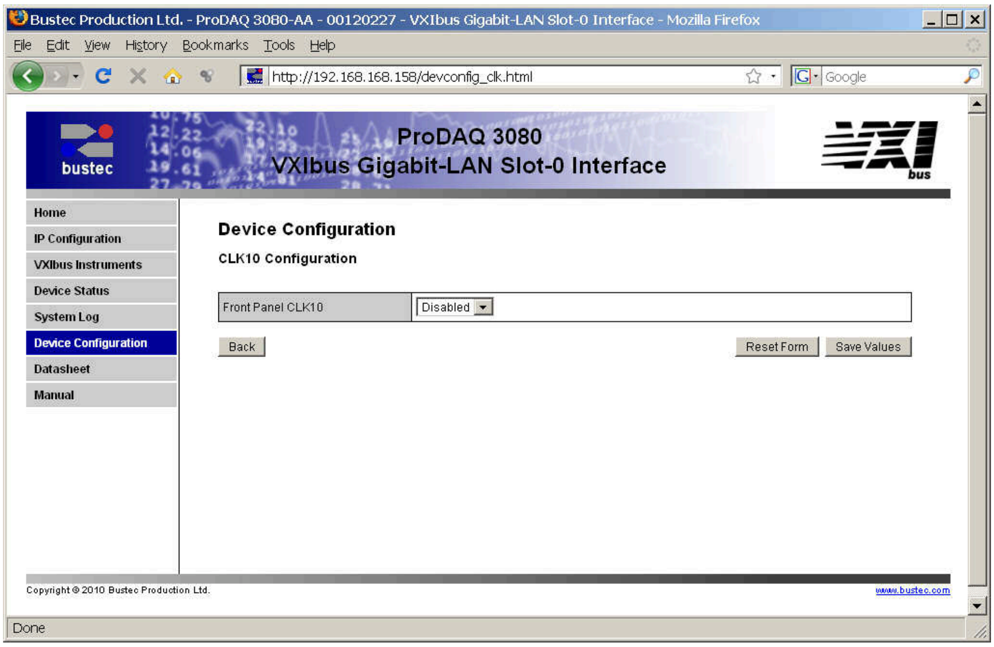

Figure 25 - CLK10 Configuration Page

3.6.6 Reboot Device

If for any reason you need to reboot the ProDAQ 3080 remotely, you can use the button “Reboot Device” in the “Device Configuration” page. To avoid accidental usage of this feature, selecting the button will cause a verification dialog to pop-up before the actual reboot starts.

Note:

Please allow sufficient time for the device to reboot before trying to access it again. Please note as well that depending on your IP and network configuration the device may use a different IP address after reboot (e.g. DHCP).

3.6.7 Firmware Update

To update the firmware on the ProDAQ 3080, use the “Update Firmware” button on the “Device Configuration” page.

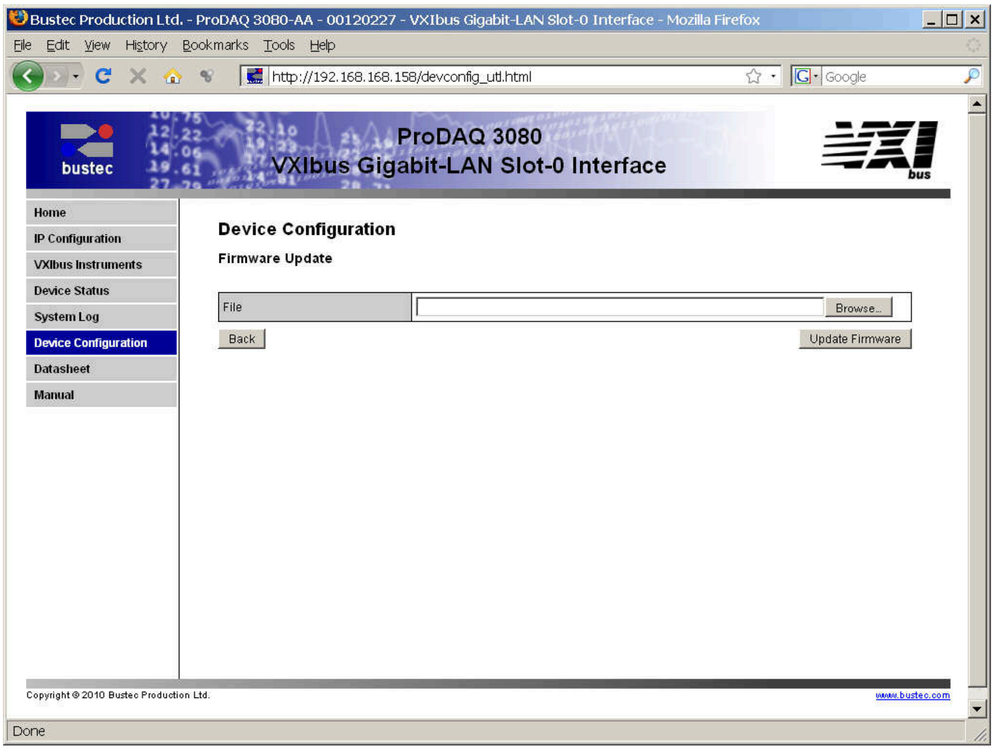

Figure 26 – Firmware Update Page

First save the file containing the new image on your local host. Press the "Browse..." to open the file upload dialog, which allows you to browse through your file system and select the file to upload. Once the correct file is selected, press the "Update Firmware" button. The upload progress and the programming progress will be displayed by a progress bar below the file selection control.

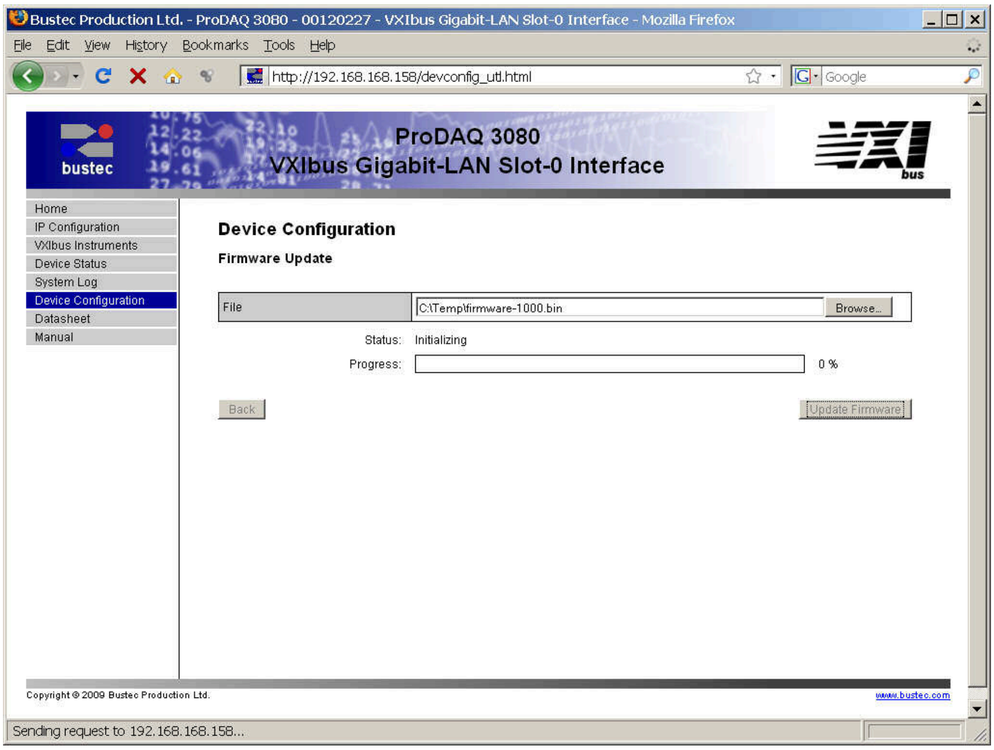

Figure 27 - Firmware Upload Progress

During the upload and programming, do not navigate away from the page by using the browser controls. Any interruption of the update process might render the ProDAQ 3080 unusable.

WARNING

Depending on your connection speed uploading and programming a new firmware image may take several minutes. To safely complete the process, do not navigate away from the page and do not interrupt the connection to the ProDAQ 3080 or power-cycle the mainframe.

3.7 Datasheet and Manual Pages

The “Datasheet” and “Manual” pages allow you to view or download the ProDAQ 3080 datasheet and user manual.

This page was intentionally left blank.

Chapter 4 - Remote Operation

The ProDAQ 3080 Gigabit LAN Slot-0 Interface features a VXI-11 RPC server, which allows the access from remote hosts via the VISA library. This access can be done in two ways, either by accessing the VXIbus instruments separately as TCP/IP instruments or by mapping the ProDAQ 3080 into the remote VISA configuration as a standard VXIbus interface.

4.1 TCP/IP Instrument Access

To access the VXIbus instruments installed in the same mainframe as the ProDAQ 3080 interface, you will need to use resource strings in the format

```
TCPIP[board]::<host address>::<interface>,<logical address>[::INSTR]
```

where [board] is the optional index of the LAN interface devices (as default, device 0 is used); <host address> specifies the host name or IP number of the ProDAQ 3080 interface; <interface> specifies which interface on the ProDAQ 3080 to use (currently only "vxi0" is supported) and <logical address> specifies the logical address of the VXIbus instruments to access. The specification "::INSTR" is optional.

Example: If the ProDAQ 3080 is configured to use IP address 192.168.1.80 and is installed in the same mainframe as a VXIbus device configured for using logical address 2, access to this device can be gained by using the open statement

```
status = viOpen (rm_session,
                 "TCPIP::192.168.1.80::vxi0,2::INSTR",
                 VI_NULL, VI_NULL, &instr_session);
```

Hence, as the VXI-11 standard allows only for read/write RPC messages, only message based VXIbus instruments can be operated in this way.

4.2 Mapped Interface Access

To gain access to all VXIbus instruments via the ProDAQ 3080 Gigabit LAN Slot-0 Interface, it is recommended to map the ProDAQ 3080 as a standard VXIbus interface onto the host system.

To do so, select the “VISA Configuration Utility” (“Start” ➔ “VXIPNP” ➔ “VISA Configuration Utility”) from the VXI*plug*&*play* program group created during the installation of the VISA library. This will start the configuration tool for the VISA library and attached hardware interfaces.

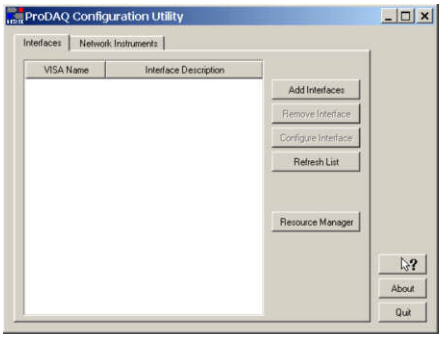

Figure 28 - VISA Configuration Utility

To add a new interface, select “Add Interfaces”. A new dialog “Available Interfaces” is shown with a list of unconfigured devices found in the host system. To map a remote interface, select the "Map Network Interface" button at the bottom.

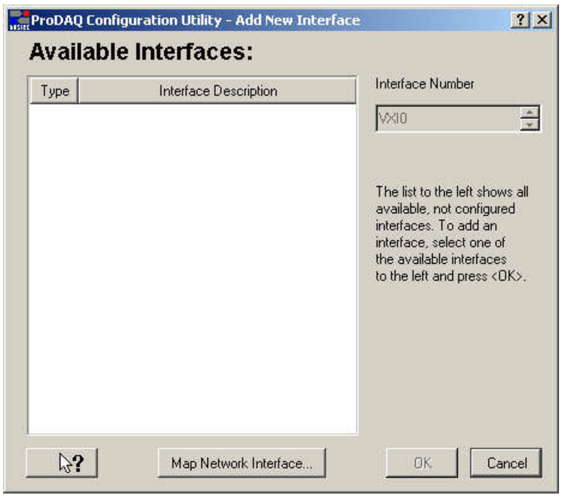

Figure 29 - Add New Interface Dialog

In the "Add Network Interface" dialog you can specify the network address of the remote interface and the local interface on the remote server to use:

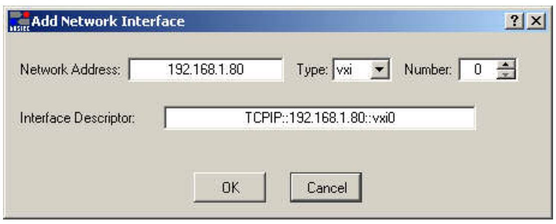

Figure 30 - Add Network Interface Dialog

To map the ProDAQ 3080 as a VXIbus interface, select type "VXI" and the interface number "0". After selecting "Ok", the remote interface will be visible in the list of available interfaces:

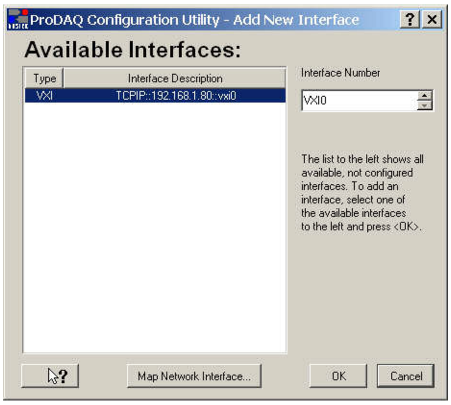

Figure 31 - Updated Available Interfaces List

Now you only need to select the interface by clicking on its entry in the list and select an interface number on the right side to which it should be mapped to on this computer. After selecting "Ok", the device is visible in the list of configured interfaces and can be used by any program running on this computer using the VISA library for instrument I/O:

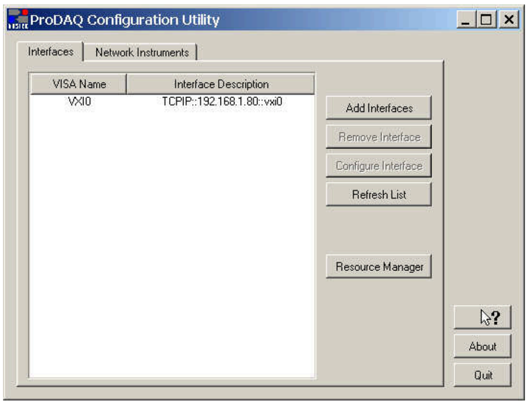

Figure 32 - Updated Configured Interfaces List

As the remote interface is now mapped as a standard VXIbus interface onto the computer, the resource manager need to run to retrieve the instrument configuration from the remote host.

To run the resource manager, select “VXIbus Resource Manager” from the VXI*plug*&*play* program group in the start menu (“Start” ➔ “VXIPNP” ➔”VXI Resource Manager”). During the identification phase, it will connect to the remote interface and retrieve the configuration information stored by the embedded resource manager of the remote interface.

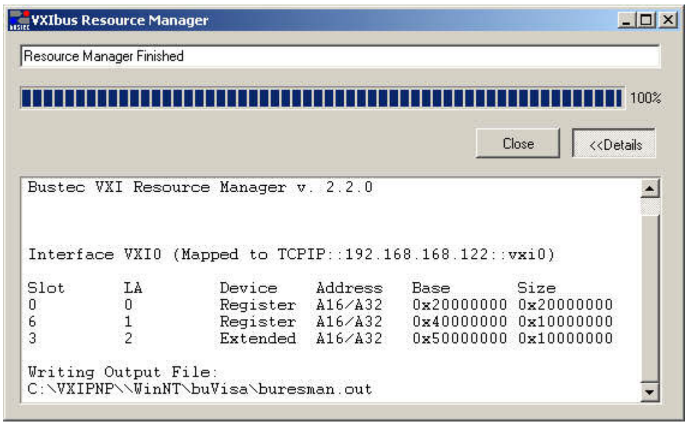

Figure 33 - Resource Manager

Note

The VISA library is a shared library that initializes itself when it is first loaded by an application. Applications started while the VISA library is already loaded just share this configuration. Only when all applications using the VISA library are stopped, it will be unloaded by the system. Therefore all applications using the VISA library must be closed before running the resource manager or using the VISA configuration utility. Take special care while using integrated development environments, they will keep the VISA library loaded even when the application developed in them was stopped.

4.3 The VISA Assistant

The VISA Assistant is an interactive tool, which allows executing VISA commands without programming. To run the VISA Assistant, select “VISA Assistant” from the VXI*plug*&*play* program group in the start menu (“Start” ➔ “VXIPNP” ➔”VISA Assistant”).

The main window of the Visa Assistant shows a list of all VISA resources in the system:

Figure 34 - The VISA Assistant

On selecting one by double-clicking on its entry, the VISA Assistant opens a VISA session for that device in a separate window:

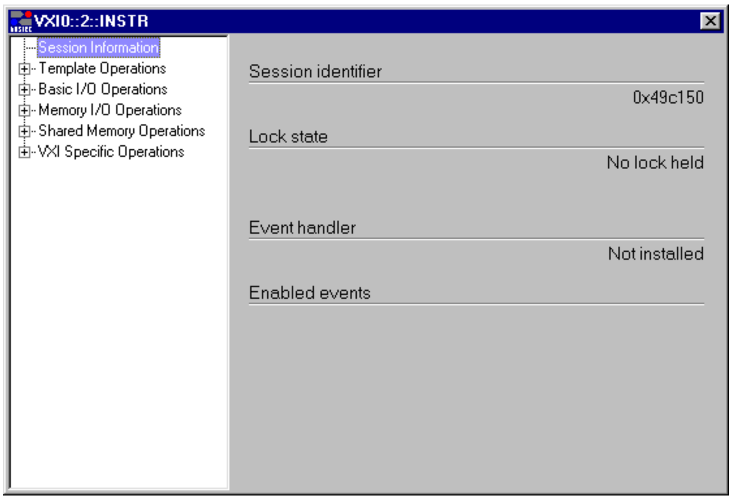

Figure 35 - VISA Assistant Session Window

In the treeview control on the left hand side you have now access to information about the session and the VISA functions possible for the resource.

The functions available are divided into five groups:

- Template Operations
- Basic I/O Operations
- Memory I/O Operations
- Shared Memory Operations
- VXI Specific Operations

Not all operations are available for all types of devices, so depending on the device type, the treeview control might not list all the possibilities discussed here.

4.3.1 Template Operations

The VISA standard implements a template of standard services for a resource. The functions in this group provide access to those services. The services available include attribute operations, asynchronous operation control, resource access control and event operations.

As an example, the function viGetAttribute allows to retrieve the values for attributes defined for a resource. Selecting the function in the treeview control on the left hand side (click on "Template Operations", then on "viGetAttribute") allows you to control the parameters for the function in a dialog on the right hand side of the session window:

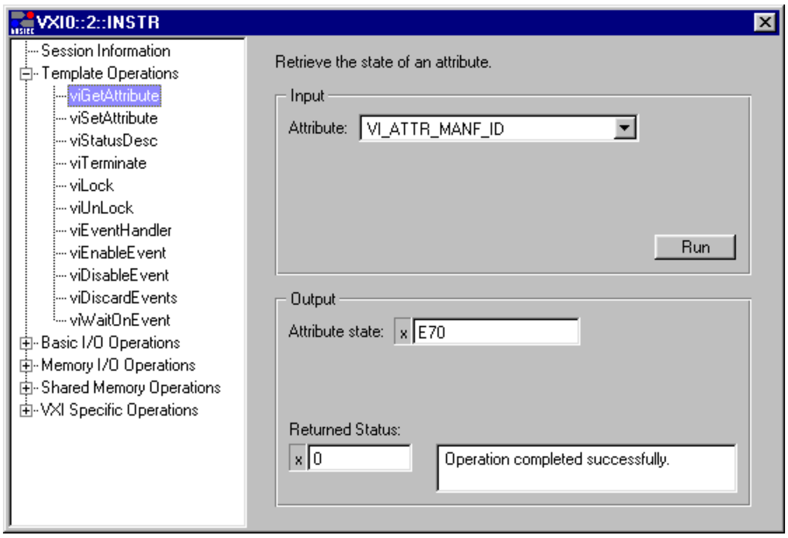

Figure 36 - Using a template operation

Select one of the attributes to retrieve in the "Attribute" control in the "Input" section and press "Run". The "Output" section will show the current value of the attribute in the control "Attribute state", if the operation was successful, and the returned status of the function.

4.3.2 Basic I/O Operations

The basic I/O operations will allow the user to send commands to a device and read back its answer, to trigger the device or read its status.

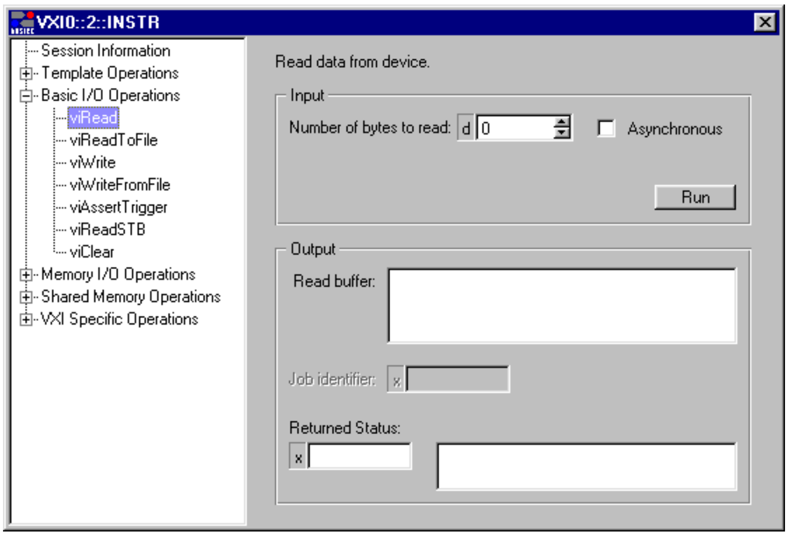

Figure 37 - Using a basic I/O operation

As an example, you can use the viRead function to read data or a message from the device. To do so, just specify the maximum number of bytes to read from the device and press “Run”. As before, the VISA Assistant will show the message read as well as the returned status of the operation.

4.3.3 Memory I/O Operations

The memory I/O operations consist of High- and Low-Level Access services. The High-Level Access Services allow register-level access to devices that support direct memory access. They encapsulate most of the code required to perform the access, such as window mapping, address translation and error checking. The Low-Level Access Services are similar in purpose, but are implemented without the software overhead of the High-Level Services.

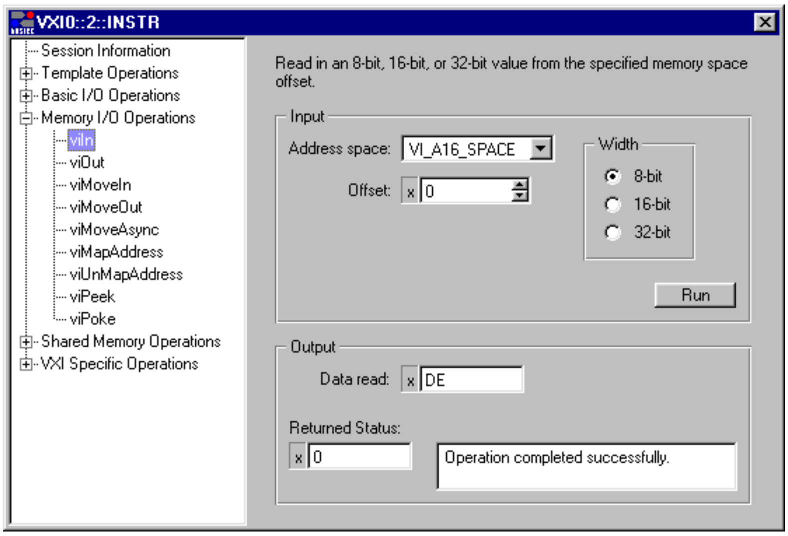

Figure 38 - Memory I/O Operations

Figure 38 shows an example of the high-level access services. In the “Input” section the user can select an address space, an offset and a transfer width. By pressing “Run”, on of the functions viIn8, viIn16 or viIn32 (depending on the access width) are executed and the result is shown in the “Output” section of the dialog along with the returned status.

The high-level functions viMoveIn, viMoveOut and viMoveAsync will move blocks of data. As with the functions viIn8, vIn16, viIn32, viOut8, viOut16 and viOut32, the “Input” section will allow you to enter an address space, an offset and a transfer width. Additionally a length parameter will define the number of elements to transfer.

The low-level access services viMapAddress, viUnmapAddress, viPeek and viPoke need to be used together. First a memory mapping must be established by using the function viMapAddress, then viPeek and viPoke can be used to access the mapped register space, and viUnmapAddress must be used to undo the memory mapping.

4.3.4 Shared Memory Operations

Shared memory operations allow allocating memory space on the device to be used exclusively by the session allocating it. Figure 39 shows an example of the shared memory operations.

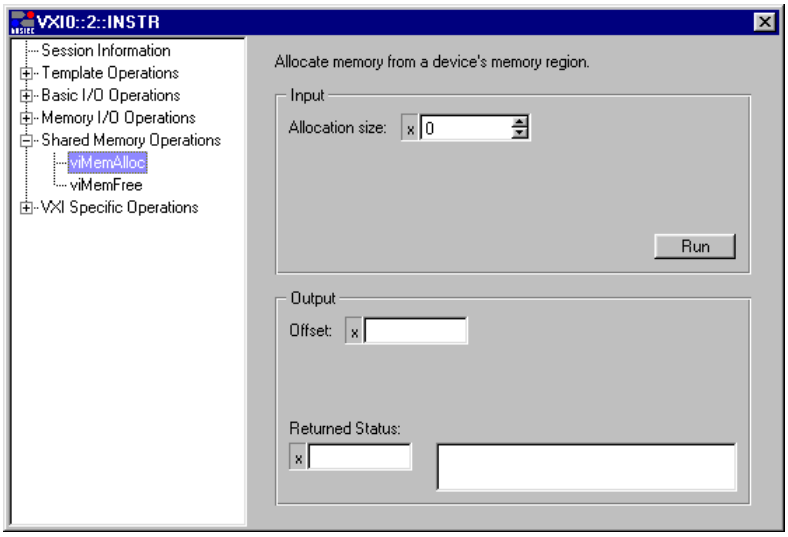

Figure 39 - Shared Memory Operations

4.3.5 VXI Specific Operations

VXI Specific Operations are those operations, which were implemented to deal with special circumstances you can find only on controller and instruments using the VXIbus to communicate. The example shows an operation, which can be found only for backplane resources of VXIbus mainframes.

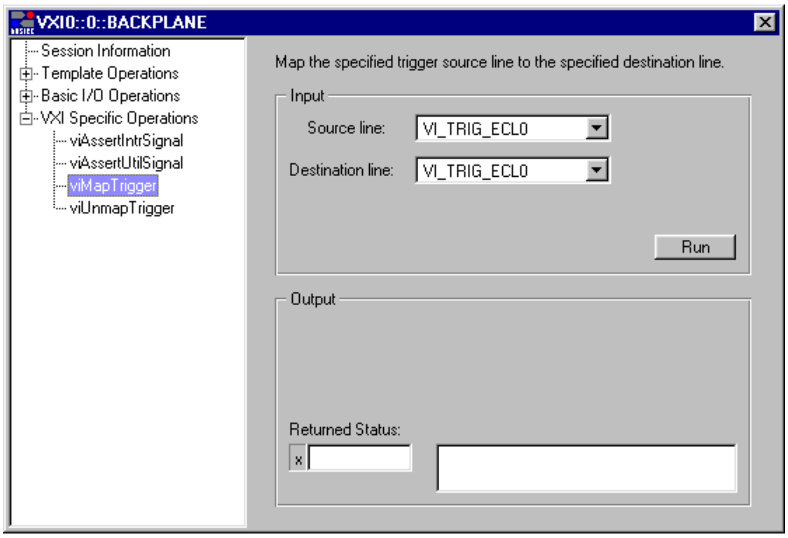

Figure 40 - VXI Specific Operations

The functions viMapTrigger and viUnmapTrigger enable you to route a trigger signal from a front panel input to one of the VXIbus trigger lines (only for VXIbus controller supporting this feature). In the “Input” section you can select a source trigger line, which should be mapped to a destination trigger line. As in the other examples, pressing “Run” will execute the function and display the result in the “Output” section.

Note

For more information about the VISA functions and their parameter, refer to the VXIplug&play Systems Alliance document “VPP-4.3: The VISA Library”.

This page was intentionally left blank.

Chapter 5 - Programming VXI Devices

This chapter shows how to use the ProDAQ 3080 Embedded VXIbus Slot-0 Controller and the Bustec VISA library to program VXI instruments. The following examples assume that the ProDAQ 3080 was mapped as a VXI interface on the host.

5.1 Connecting to a Device

An application using the VISA library to communicate with the instrument needs to open a session for the resource it wants to use. A resource might be a physical resource as for example a VXI instrument or virtual resources like the backplane or the resource manager. The session will handle all accesses, attributes and services for the particular resource.

```
#include <visa.h>

main (int argc, char **argv)
{
    ViStatus status;
    ViSession rm_session;
    ViSession instr_session;
    ViChar descr[256];

    /* open a session to the resource manager */
    if ((status = viOpenDefaultRM (&rm_session)) != VI_SUCCESS)
①   {
        viStatusDesc (rm_session, status, descr);

        if (status > VI_SUCCESS)
            printf ("VISA WARNING: viOpenDefaultRM returned status %08x (%s)\n",
                status, descr);
        else
        {
            printf ("VISA ERROR: viOpenDefaultRM returned status %08x (%s)\n",
                status, descr);
            return status;
        }
    }

    /* open a session to the instrument */
②   if ((status = viOpen (rm_session, "VXI0::2::INSTR",
                                    VI_NULL, VI_NULL, &instr_session)) != VI_SUCCESS)
    {
        viStatusDesc (instr_session, status, descr);

        if (status > VI_SUCCESS)
            printf ("VISA WARNING: viOpen returned status %08x (%s)\n",
                status, descr);
        else
        {
            printf ("VISA ERROR: viOpen returned status %08x (%s)\n",
                status, descr)
            return status;
        }
    }

    /* accessing the instrument */

    /* close the sessions to the instrument and the resource manager */
③   viClose (instr_session);
    viClose (rm_session);
}
```

Figure 41 - Opening a VISA Session

The example shown in Figure 41 contains all necessary steps to connect to a device using VISA functions. The first step in a program, which uses the VISA library, is always to open a session to the default resource manager (①). It provides connectivity to all VISA resources registered with it and gives applications control and access to individual resources.
The next step is to open a session to the instrument or multiple sessions to multiple instruments (②). The resource name used is a combination of interface type and number, logical address of the VXI device, and a device type:

VXI 0 :: 2 :: INSTR

Interface Type

Interface Number

Logical Address

Device Type

The interface type for the ProDAQ 3080 Slot-0 Controller is always “VXI”. The interface number is the number, which was assigned to the particular 3080 by using the VISA configuration utility (see 4.2: Mapped Interface Access). The logical address of a VXI device is defined either statically by setting its logical address switch, or dynamically during runtime by the resource manager. If the resource manager assigned the address dynamically, the actual assignment can be found in the output file of the resource manager. The device type for VXI instruments is always “INSTR”.

Note

When running the above example, please make sure that the logical address used in it matches the logical address setting of the instrument you want to connect to.

Note

Before you can use the above example to connect to your device, you must run the VXI Resource Manager.

5.2 Programming Register-based Devices

Register-based devices are devices implementing a set of registers in A16 and often in A24 or A32. Programming register-based devices is done by reading and writing these registers to change their contents, either by bit, in groups of bits or in whole.

5.2.1 Accessing Registers

To access single registers, the VISA library offers two groups of functions. The first group, viIn8, viIn16, viIn32, viOut8, viOut16, viOut32, provides a standardized, single word access to a device register in A16, A24 or A32 space. Figure 42 shows an example of a function reading a value from a device register (①), modifying the value read and writing it back (②). The driver for the ProDAQ 3080 will automatically take care about byte ordering, i.e. it will swap the words to be read or written between the little-endian host byte ordering your PC is using to the big-endian byte ordering used on the VXIbus.


```
ViStatus function rmw_register (ViSession instr_session, ViBusAddress offset, ViUInt16 mod)
{
    ViStatus status;
    ViChar descr[256];
    ViUInt16 value;

①   if ((status = viIn16 (instr_session, VI_A16_SPACE, offset, &value) != VI_SUCCESS)
    {
        viStatusDesc (instr_session, status, descr);

        if (status > VI_SUCCESS)
            printf (“VISA WARNING: viIn16 returned status %08x (%s)\n”, status, descr);
        else
        {
            printf (“VISA ERROR: viIn16 returned status %08x (%s)\n”, status, descr);
            return status;
        }
    }

    value = value | mod;

②   if ((status = viOut16 (instr_session, VI_A16_SPACE, offset, value) != VI_SUCCESS)
    {
        viStatusDesc (instr_session, status, descr);

        if (status > VI_SUCCESS)
            printf (“VISA WARNING: viOut16 returned status %08x (%s)\n”, status, descr);
        else
        {
            printf (“VISA ERROR: viOut16 returned status %08x (%s)\n”, status, descr);
            return status;
        }
    }

    return VI_SUCCESS;
}
```

Figure 42 - Memory-based I/O

The second group of functions is intended to map a register range into the memory of the host and accessing it directly. Because this ability is architecture and system dependent, the VISA standard foresees an attribute, which allows determining whether the range could be physically mapped or the system architecture does not allow it. Depending on the value of the attribute VI_ATTR_WIN_ACCESS, the range mapped can be directly accessed (e.g. by using a C-style pointer), or the functions viPeek8, viPeek16, viPeek32, viPoke8, viPoke16 and viPoke32 must be used to access registers in the mapped range. Figure 43 shows the same function as in Figure 42, this time implemented with memory mapping functions.


```
ViStatus function rmw_register (ViSession instr_session, ViBusAddress offset, ViUInt16 mod)
{
    ViStatus status;
    ViChar descr[256];
    ViAddr address;
    ViUInt16 win_access;
    ViUInt16 value;

①   if ((status = viMapAddress (instr_session, VI_A32_SPACE, offset,
                     sizeof (ViUInt16), VI_FALSE,  (ViAddr) 0, &address)) != VI_SUCCESS)
    {
        viStatusDesc (instr_session, status, descr);

        if (status > VI_SUCCESS)
            printf (“VISA WARNING: viMapAddress returned status %08x (%s)\n”,
                     status, descr);
        else
        {
            printf (“VISA ERROR: viMapAddress returned status %08x (%s)\n”,
                     status, descr);
            return status;
        }
    }

②   if ((status = viGetAttribute (instr_session,
                              VI_ATTR_WIN_ACCESS, &win_access)) != VI_SUCCESS)
    {
        viStatusDesc (instr_session, status, descr);

        if (status > VI_SUCCESS)
            printf (“VISA WARNING: viGetAttribute returned status %08x (%s)\n”,
                     status, descr);
        else
        {
            printf (“VISA ERROR: viGetAttribute returned status %08x (%s)\n”,
                     status, descr);
            return status;
        }
    }

    if (win_access == VI_DEREF_ADDR)
    {
        /* allowed to use pointer or similar */
        value = *((ViUInt16 *) address);
③       value = value | mod;
        *((ViUInt16 *) address) = value;
    }
    else if (win_access == VI_USE_OPERS)
    {
        /* use functions to access memory */
        viPeek16 (instr_session, address, &value);
④       value = value | mod;
        viPoke16 (instr_session, address, value);
    }

⑤   if ((status = viUnmapAddress (instr_session) != VI_SUCCESS)
    {
        viStatusDesc (instr_session, status, descr);

        if (status > VI_SUCCESS)
            printf (“VISA WARNING: viUnmapAddress returned status %08x (%s)\n”,
                     status, descr);
        else
        {
            printf (“VISA ERROR: viUnmapAddress returned status %08x (%s)\n”,
                     status, descr);
            return status;
        }
    }

    return VI_SUCCESS;
}
```

Figure 43 - Register I/O using memory mapping

In the above example, the function viMapAddress is used to map a register range starting with *offset* and extending over the size of the register into the memory of the host (①). If this is successful, the attribute "VI_ATTR_WIN_ACCESS" is checked to see whether the controller was able to map the address range physically into the memory space of the controller, or whether the mapping was done only logically (②). If the mapping was done physically, the application is allowed to use the address, the register range is mapped to, as if it is accessing its own memory. So for example C-style pointers may be used to change the register value (③). If the mapping was done only logically, the application need to use the functions viPeek and viPoke provided by the VISA library to access the mapped register range (④). The VISA library will use the stored values for the mapped offset and range to calculate the physical address and execute a single access in the same way as internally done for the high-level functions. The function viUnmapAddress must be used to undo the mapping of the register range (⑤). Only one mapping per session is allowed by the VISA standard. Please not that the functions viPeek and viPoke will work in both cases (VI_ATTR_WIN_ACCESS equal to VI_DEREF_ADDR or equal to VI_USE_OPERS), but will introduce a slightly higher overhead then using direct access if possible.

5.2.2 Moving Blocks of Data

To move blocks of data between an instruments memory and the host memory, the VISA library implements the functions viMoveIn and viMoveOut for different transfer sizes. In addition a number of attributes can be used to define the type of transfer performed on the VXIbus.

```
#include <visa.h>

/* buffer used to store data from the instrument */
ViUInt16 data[1024];

main (int argc, char **argv)
{
    ViStatus status;
    ViSession rm_session;
    ViSession instr_session;
    ViChar descr[256];
    ViUInt16 value;

    /* open a session to the resource manager and instrument
     * as shown in Figure 41 - Opening a VISA Session (not shown here) */
     . . . .

    /* now move a block of 16-bit data from the instrument to the buffer */
    if ((status = viMoveIn16 (instr_session,
                              VI_A32_SPACE, MEM_START, 1024, data) != VI_SUCCESS)
    {
        viStatusDesc (instr_session, status, descr);

        if (status > VI_SUCCESS)
            printf ("VISA WARNING: viMoveIn16 returned status %08x (%s)\n", status, descr);
        else
        {
            printf ("VISA ERROR: viMoveIn16 returned status %08x (%s)\n", status, descr);
            return status;
        }
    }

    /* close the sessions as shown in Figure 41 - Opening a VISA Session */
    . . . . .
}
```

Figure 44 - Moving a Block of Data

For each move, one or several packets of data are moved over the VXIbus to the ProDAQ 3047. The type of transfer used on the VXIbus depends on the value of several attributes:

VI_ATTR_SRC_PRIV for data moved from a VXIbus instrument to the host

VI_ATTR_DEST_PRIV for data moved from the host to a VXIbus instrument

Only if the value of those attributes are set correctly prior to moving the data via viMoveIn or viMoveOut, a block transfer on the VXIbus will take place. The following table shows the type of transfers performed by the viMoveIn, viMoveOut and viMove functions for the different values of the attributes:

Settings		**Resulting Transfer**			
Attribute	Address Space	Privilege	Data/Program	Block Transfer	AM(hex)
VI_DATA_PRIV	VI_A16_SPACE	Supervisory	-	-	2D
	VI_A24_SPACE	Supervisory	Data	-	3D
	VI_A32_SPACE	Supervisory	Data	-	0D
VI_DATA_NPRIV	VI_A16_SPACE	Non-priv.	-	-	29
	VI_A24_SPACE	Non-priv.	Data	-	39
	VI_A32_SPACE	Non-priv.	Data	-	09
VI_PROG_PRIV	VI_A16_SPACE	Supervisory	-	-	2D
	VI_A24_SPACE	Supervisory	Program	-	3E
	VI_A32_SPACE	Supervisory	Program	-	0E
VI_PROG_NPRIV	VI_A16_SPACE	Non-priv.	-	-	29
	VI_A24_SPACE	Non-priv.	Program	-	3A
	VI_A32_SPACE	Non-priv.	Program	-	0A
VI_BLCK_PRIV	VI_A16_SPACE	Supervisory	-	-	2D
	VI_A24_SPACE	Supervisory	-	BLT	3F
	VI_A32_SPACE	Supervisory	-	BLT	0F
VI_BLCK_NPRIV	VI_A16_SPACE	Non-priv.	-	-	29
	VI_A24_SPACE	Non-priv.	-	BLT	3B
	VI_A32_SPACE	Non-priv.	-	BLT	0B
VI_D64_PRIV	VI_A16_SPACE	Supervisory	-	-	2D
	VI_A24_SPACE	Supervisory	-	MBLT	3C
	VI_A32_SPACE	Supervisory	-	MBLT	0C
VI_D64_NPRIV	VI_A16_SPACE	Non-priv.	-	-	29
	VI_A24_SPACE	Non-priv.	-	MBLT	38
	VI_A32_SPACE	Non-priv.	-	MBLT	08
VI_2eVME_PRIV	VI_A32_SPACE	-	-	2eVME	20 / 01
VI_2eVME_NPRIV	VI_A32_SPACE	-	-	2eVME	20 / 01

Figure 45 - VXIbus transfer types

Block transfers are performed on the VXIbus only if the correct attribute (VI_ATTR_SRC_PRIV or VI_ATTR_DEST_PRIV, depending on the direction) is set to one of the types VI_BLCK_PRIV, VI_BLCK_NPRIV, VI_D64_PRIV or VI_D64_NPRIV. The data width of the performed transfer depends on the viMoveXX function used, except for the case that the attribute is set to VI_D64_PRIV, VI_D64_NPRIV, VI_2eVME_PRIV or VI_2eVME_NPRIV in which case a D64 MBLT resp. 2eVME transfer is performed (viMoveIn32 and viMoveOut32 only).

Please note that the attributes VI_2eVME_PRIV and VI_2eVME_NPRIV are temporary assigned by Bustec to allow the usage of the 2eVME block transfers in compliance with the VXIbus Standard revision 3.0. The 2eVME block transfers are not yet part of the VISA standard VPP-32 and may be named differently once they are.


```
#include <visa.h>

ViUInt16 data[1024];       /* buffer used to store data */

main (int argc, char **argv)
{
    ViStatus status;
    ViSession rm_session;
    ViSession instr_session;
    ViChar descr[256];
    ViUInt16 value;

    /* open a session to the resource manager and instrument
     * as shown in Figure 41 - Opening a VISA Session (not shown here) */

    /****************************************************************************/
    /*  Perform a 16-bit wide block transfer from a VXIbus instrument to the host   */
    /****************************************************************************/

    /* set the correct attribute - VI_ATTR_SRC_PRIV for moving data IN */
    if ((status = viSetAttribute (instr_session,
                                   VI_ATTR_SRC_PRIV, VI_BLK_PRIV)) != VI_SUCCESS)
    {
        /* handle errors or warnings (not shown here) */
    }

    /* now move a block of 16-bit data from the instrument to the buffer */
    if ((status = viMoveIn16 (instr_session,
                               VI_A32_SPACE, MEM_START, 1024, data) != VI_SUCCESS)
    {
        /* handle errors or warnings (not shown here) */
    }

    /****************************************************************************/
    /*  Perform a 32-bit wide block transfer from the host to a VXIbus instrument   */
    /****************************************************************************/

    /* set the correct attribute - VI_ATTR_DEST_PRIV for moving data OUT */
    if ((status = viSetAttribute (instr_session,
                                   VI_ATTR_DEST_PRIV, VI_BLK_PRIV)) != VI_SUCCESS)
    {
        /* handle errors or warnings (not shown here) */
    }

    /* now move a block of 32-bit data from the instrument to the buffer */
    if ((status = viMoveOut32 (instr_session,
                               VI_A32_SPACE, MEM_START, 1024, data) != VI_SUCCESS)
    {
        /* handle errors or warnings (not shown here) */
    }

    /****************************************************************************/
    /*  Perform a 64-bit wide block transfer from the host to a VXIbus instrument   */
    /****************************************************************************/

    /* set the correct attribute - VI_ATTR_DEST_PRIV for moving data OUT */
    if ((status = viSetAttribute (instr_session,
                                   VI_ATTR_DEST_PRIV, VI_D64_PRIV)) != VI_SUCCESS)
    {
        /* handle errors or warnings (not shown here) */
    }

    /* now move a block of 64-bit data from the instrument to the buffer */
    if ((status = viMoveOut32 (instr_session,
                               VI_A32_SPACE, MEM_START, 1024, data) != VI_SUCCESS)
    {
        /* handle errors or warnings (not shown here) */
    }

    /* close the sessions as shown in Figure 41 - Opening a VISA Session */
}
```

Figure 46 - Performing VXIbus Block Transfers

5.3 Programming Message-based Devices

Message-based VXIbus devices implement the word serial protocol to communicate with the application. Programming is done by sending ASCII messages to the device and reading its answer.

5.3.1 Writing and Reading Messages

The basic functions to write and read messages to/from devices are the two functions viRead and viWrite. They implement the word serial protocol for message based devices, but on a very basic level. The user needs to build his message and use viWrite to send it to the device. Then he uses viRead to receive the message sent back. The message received might consist of strings, numbers and formatting characters and he will need to interpret this message. To avoid some of these steps, a couple of higher level functions were implemented in the VISA library.

```
#include <visa.h>

main (int argc, char **argv)
{
     ViStatus status;
     ViSession rm_session;
     ViSession instr_session;
     ViChar descr[256];

     /* open a session to the resource manager */
     if ((status = viOpenDefaultRM (&rm_session)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* open a session to the instrument */
     if ((status = viOpen (rm_session, "VXI0::2::INSTR",
                                        VI_NULL, VI_NULL, &instr_session)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* reset the device */
①    if ((status = viPrintf (vi, "*RST\n")) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* ask the device for its identification */
②    if ((status = viPrintf (vi, "*IDN?\n")) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* read the identification sent back */
③    if ((status = viScanf (vi, "%256t", descr)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }
     printf ("Device Identification: %s\n", descr);

     /* close the sessions to the instrument and the resource manager */
     viClose (instr_session);
     viClose (rm_session);
}
```

Figure 47 - Reading the Device Identification

The functions ViPrintf and viScanf use a C-style formatting string to format and scan messages send to and read from the device, freeing the user from the separate steps necessary to do so, if using the lower level function viWrite and viRead. Furthermore the functions implement an extended set of formatting styles specially shaped towards instrument communication.

In the above example the function viPrintf is used to send two messages to the device, first a command to reset the device (①), then a request to send back its identification string (②). viPrinf uses the format string together with the other arguments passed to it to build a message string in a local buffer and then it calls viWrite to send this message to the device.

The example program reads the identification using the function viScanf (③). ViScanf allocates a local buffer, calls the function viRead to receive the message form the device and then it parses the message using the formatting supplied by the format string. In the example the format code “%t” together with a size modifier is used, telling viScanf to expect a string to be returned in the message, and to copy a maximum of 256 characters into the buffer supplied.

The VISA standard support a wide range of formatted I/O services like the viPrintf/viScanf functions shown in the example. Please refer to the VISA standard document “VXI*plug*&*play* Systems Alliance VPP-4.3: The VISA library” for a complete list.

5.4 Optimizing Data Throughput

To optimize you programs to achieve the maximum data throughput, please keep the following in mind:

- Use the functions viMove, viMoveIn or viMoveOut instead of single read and write commands for devices and register ranges, where this is possible.
- Use the attributes VI_ATTR_SRC_PRIV and VI_ATTR_DEST_PRIV to specify block transfer privileges for devices where this is possible.
- Use 32-bit or 64-bit moves, whenever possible.

5.5 Using VXIbus and Front Panel Trigger Lines

One feature, that differs the VXIbus from other busses, is its ability to use trigger signals to communicate with instruments in real-time, to share clock signals, etc. The VISA library implements functions to control those trigger lines from your application.

5.5.1 Using VXIbus Trigger Lines

The VISA standard implements the function viAssertTrigger together with the attribute VI_ATTR_TRIG_ID to assert and de-assert trigger lines on the VXIbus or sending the word serial trigger command to message-based devices.

```
#include <visa.h>

main (int argc, char **argv)
{
     ViStatus status;
     ViSession rm_session;
     ViSession instr_session;
     ViChar descr[256];

     /* open a session to the resource manager */
     if ((status = viOpenDefaultRM (&rm_session)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* open a session to the instrument */
     if ((status = viOpen (rm_session, "VXI0::2::INSTR",
                                        VI_NULL, VI_NULL, &instr_session)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* defining the trigger line to use */
     if ((status = viSetAttribute (instr_session,
①                                  VI_ATTR_TRIG_ID, VI_TRIG_TTL0)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* send a trigger pulse to the device */
②    if ((status = viAssertTrigger (instr_session, VI_TRIG_PROT_SYNC)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* close the sessions to the instrument and the resource manager */
     viClose (instr_session);
     viClose (rm_session);
}
```

Figure 48 - Sending a Trigger Pulse

Figure 48 shows an example for sending a trigger pulse to a device. The function viSetAttribute is used (①) to set the attribute VI_ATTR_TRIG_ID to select the trigger line. In general the trigger ID can be set to VI_TRIG_TTL0 to VI_TRIG_TTL7, VI_TRIG_ECL0/VI_TRIG_ECL1 or VI_TRIG_SW. For the setting VI_TRIG_SW, the device is sent the word serial trigger command, the other settings correspond to the VXIbus trigger lines TTL0-TTL7 and ECL0/ECL1.
To send the trigger, the function viAssertTrigger is used in the example (②) with the "protocol" argument set to VI_PROT_DEFAULT. The interpretation of this argument depends on the value, the attribute VI_ATTR_TRIG_ID is set to. For software triggers, the only valid protocol is VI_PROT_DEFAULT. For hardware triggers, the protocols VI_PROT_DEFAULT or VI_PROT_SYNC will generate a trigger pulse on the specified line, while VI_PROT_ON and VI_PROT_OFF let you explicitly assert and de-assert the trigger line.

5.5.2 Using Front-Panel Trigger Lines

The ProDAQ 3080 supports a front-panel trigger input and output, which can be mapped to the VXIbus trigger lines. For this purpose, as for querying and manipulating other VXIbus backplane specific lines, the VISA standard implements a special resource. It encapsulates the VXI-defined operations and properties of the backplane in a VXIbus

system. It lets a controller query and manipulate specific lines on a specific mainframe in a given VXI system. Services are provided to map, unmap, assert, and receive hardware triggers, and also to assert various utility and interrupt signals.

The resource descriptor used for the backplane resource is again a combination of interface type and number, logical address of the VXI device, and the device type BACKPLANE:

VXI 0 :: 0 :: BACKPLANE

Interface Type

Interface Number

Logical Address

Device Type

As before, the interface type is always “VXI”. The interface number depends on the assignment you made using the configuration utility (see 4.2: Mapped Interface Access). The logical address will be zero (0), as you will need to configure the ProDAQ 3080 for logical address zero to allow it to function as a VXIbus slot-0 controller.

Though the ProDAQ 3080 does not support the mapping of one VXIbus trigger line to another, the standard VISA functions viMapTrigger and viUnmapTrigger can be used to map the front panel trigger input to one or many of the VXIbus trigger lines as well as to map one or many VXIbus trigger lines to the front panel trigger output.
Figure 49 shows an example how to map the trigger lines to/from the front panel input and output. First a session for the backplane resource is opened (①). Then the function viMapTrigger is used to map the front panel input to the VXIbus trigger line TTL1 (②), and also to the VXIbus trigger lines ECL0 (③). This means that whenever an active trigger is detected on the front panel input of the ProDAQ 3080, both lines will be asserted. In general, when the viMapTrigger function is called multiple times with the same source trigger line and different destination trigger lines, an assertion of the source line will cause all of those destination lines to be asserted.
To map one or multiple of the VXIbus trigger lines to the front panel output, the value VI_TRIG_PANEL_OUT must be used for the destination parameter (④). As with the front panel input, multiple lines can be mapped to the front panel output. When calling viMapTrigger multiple times with the same destination line and different source lines, the destination line will be asserted when any of the source lines is asserted.

```
#include <visa.h>

main (int argc, char **argv)
{
     ViStatus status;
     ViSession rm_session;
     ViSession instr_session;
     ViChar descr[256];

     /* open a session to the resource manager */
     if ((status = viOpenDefaultRM (&rm_session)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* open a session to the instrument */
     if ((status = viOpen (rm_session, "VXI0::0::BACKPLANE",
①                                       VI_NULL, VI_NULL, &instr_session)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* mapping the front panel input to trigger line TTL1 */
     if ((status = viMapTrigger (instr_session,
②                               VI_TRIG_PANEL_IN, VI_TRIG_TTL1, VI_NULL)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* mapping the front panel input also to trigger line ECL0 */
     if ((status = viMapTrigger (instr_session,
③                               VI_TRIG_PANEL_IN, VI_TRIG_ECL0, VI_NULL)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* mapping trigger line TTL6 to the front panel output */
     if ((status = viMapTrigger (instr_session,
④                               VI_TRIG_TTL6, VI_TRIG_PANEL_OUT, VI_NULL)) != VI_SUCCESS)
     {
          /* error handling as shown in the previous examples !*/
     }

     /* close the sessions to the instrument and the resource manager */
     viClose (instr_session);
     viClose (rm_session);
}
```

Figure 49 - Mapping Trigger Lines

Bustec Production, Ltd.
World Aviation Park, Shannon, Co. Clare, Ireland
Tel: +353 (0) 61 707100, FAX: +353 (0) 61 707106